Introduction to Computer programming

Samuel Kizito

Computer programming

- Computer Programming is the process of creating computer software using a programming Language.
- It involves knowing the statements used in a programming language and how to choose and arrange those statements so that the computer performs the tasks logically

Importance of programming

- programming meets the increasing demand for computer programs.
- It provide instructions to a computer to do a specific task
- Programming provides a better understanding of how computers work.
- It helps develop thinking skills.
- It develops logical way of doing things.

- Computer programming is a lucrative job.
- It is fun.
- It is rewarding to see your ideas come out to life as a program.
- With the knowledge of programming, the user is able to evaluate software before purchasing or using one.

Limitations to programming

- Limited knowledge of a programming language
- Some programming languages are not user friendly.
- The difficulty to choose an appropriate programming language to use.

Computer program

- A Computer program is a list of logical instructions for the computer to follow in performing a task.
- A computer program is a step by step set of instructions that a computer has to work through in a logical sequence in order to carry out a particular task.

- **Programs are written by people known as Programmers.**

Features of a computer program

- Any program:

1. Has instructions to process data types including numeric and alphanumeric data.
2. Uses operations to process data which include arithmetic, relational/comparison, and logical operations.
3. performs input and output operations, must provide instructions for inputting data into memory and outputting information

4. **must count and accumulate totals for reporting purposes. The area in internal memory to record number of times an event, activity, or condition is encountered is called a counter, to record subtotal or total of certain numeric value is the accumulator**

5. **Has capability to store data temporarily in and retrieve it from internal memory for use. ie, as variables, constants, dictionaries, turples and arrays of data**

Programming Languages

- A Programming language is the vocabulary and set of grammatical rules for use by people to write instructions for the computer to perform specific tasks.
- There are many different programming languages each having a unique set of keywords (words that it understands) and a special syntax (grammar)for organising program instructions.

Examples of common programming languages

- Java
- PHP
- Python
- C
- C++
- Visual basic
- Javasript
- Ruby
- SQL

C#

Object-C

Popular programming languages

- 10 best programming languages

Levels of programming languages

Programming languages are classified into two levels: 1)The low level language is machine language or very close to machine language.

Examples of low level languages are:

- Machine languages (first generation languages),
- Assembly languages (second generation languages)

Machine language – First Generation Language (1GL)

- The machine language is low level language that writes programs using the machine code of 1s and 0s, which is directly understood by the computer.

Assembly language – Second Generation Language (2GL)

Assembly language is low level symbolic language written using **mnemonic**s (abbreviated sets of letters) or short codes that suggest their meaning and are therefore easier to remember. But must be converted to machine language before the computer can interpret it.

An example of a program code to add and store two numbers would be:

LDA A, 20: *load accumulator A with the value 20*

ADD A,10 : *add the value 10 to accumulator A*

STO B, A: *store contents of accumulator A into storage register B*

NOP : *no operation (stop here)*

Characteristics of 2GL

- Assembly language, being machine dependent, is faster and more efficient in the use of hardware than high-level programming languages.
- Assembly languages have to be translated into machine language by language translators known as assemblers for the processor to understand.
- Easier to write than machine language
- Assembly language is machine dependent.

The code is not very easy to understand, hence the introduction of high level programming languages.

Advantages of low level languages

- Very fast to execute because it is already in the language that the computer can understand.
- Require little memory resources takes up less storage space.
- No need of language translator for machine language.
- useful for writing system programs where accuracy is required

Disadvantages of low level languages

- Difficult to interpret by the programmer (requires the aid of a reference manual to interpret the meaning of each code)
- Easy to make mistakes in the sequence of 1s and 0s; replacing a 1 for a 0 can result in the wrong command/instruction being executed
- It is difficult to identify mistakes made
- Time-consuming, slow and tedious to write

- Machine dependent, e.g. one written for an IBM machine cannot work on an Apple machine.
- Makes writing of complex programs difficult.

High-level programming languages

- High level programming language is a language that is near to natural language, therefore it is machine independent and uses variables and objects, Boolean expressions, functions, loops, threads, locks which are similar to their meaning (abstraction).
- High-level languages have evolved over the years and can be grouped into five categories: Third Generation Languages (3GL), Fourth Generation Languages (4GL), Object Oriented Programming Languages (OOP), Fifth Generation Languages (5GL), Scripting Languages, and Natural Languages

- Natural languages are human languages like English or French

- High level languages are **problem oriented**.
- The high level programs are easy to write because the words and grammar of high-level languages are near to natural language.

- High-level languages are machine independent Since the syntaxes of high-level languages are standardised, the languages can be used on different computer systems.
- Programs written in a high-level language must be translated into machine language by a compiler or interpreter.

Object-Oriented programming (OOP)

- Object-oriented programming (OOP) is a programming language model organised around objects and data.
- In OOP, the programmer packages the data and the program procedures into a single unit called an object. The procedures in the object are called Operations(Methods), and the data elements are called attributes(Variables).

Advantages of high level programming languages

- High-level language programs are easy to debug
- They are machine independent. Provide programs that can be used on more than one computer.
- They are user friendly and easy to learn because they are near to natural language.
- They are flexible hence they enhance the creativity of the programmer, increasing productivity
- Allows the programmer to focus on understanding the user's needs and design the required software.
- They permit faster development of large programs.

Disadvantages of high level programming languages

- They are executed much slower than low-level programming languages
- They have to be translated into machine code before execution
- Require more memory than the low level languages

Examples of high level programming languages used

- ❑**FORTRAN** (FORmula TRAnslator)developed in the late 1950s developed to design scientific applications
- **COBOL** (Common Business Oriented Language) developed in early 1960s to develop business applications.
- **RPG** (Report Program generator) was developed in early 1960s to assist in generating reports and complex calculations.

- **BASIC** (Beginner’s All-purpose symbolic instruction code) developed in mid 1960 Basic versions include MS-BASIC, QBASIC, SmallBASIC and visual basic.
- **Pascal** was developed in the late 1960s for the purpose of teaching structured programming concepts
- **C** developed in the early 1970s to write system software
- **Ada** was developed in the late 1970s originally developed to meet the needs of embedded computer systems

- **C++** developed in the 1980s is an object-oriented language mainly to develop application software
- Note that in addition to the major languages discussed above, there are many other programming languages that have been developed such as JavaScript, and Python

Program Syntax

Program Syntax is the spelling and grammar of a programming language.
(Each program defines its own syntactical rules that control which words the computer understands, which combinations of words are meaningful, and what punctuation is necessary.)

Computer language semantics

- The Meaning of the language’s features. Semantics describes the processes a computer follows when executing a program in that specific language.

Source code

- Source code is a Program instruction written as text file by the programmer, that must be translated by a compiler or interpreter or assembler into an object code before execution.
- Source code cannot be understood by the computer until it has been translated into machine code.

Program execution

Execution is the process by which a computer system performs the instructions of a computer program.

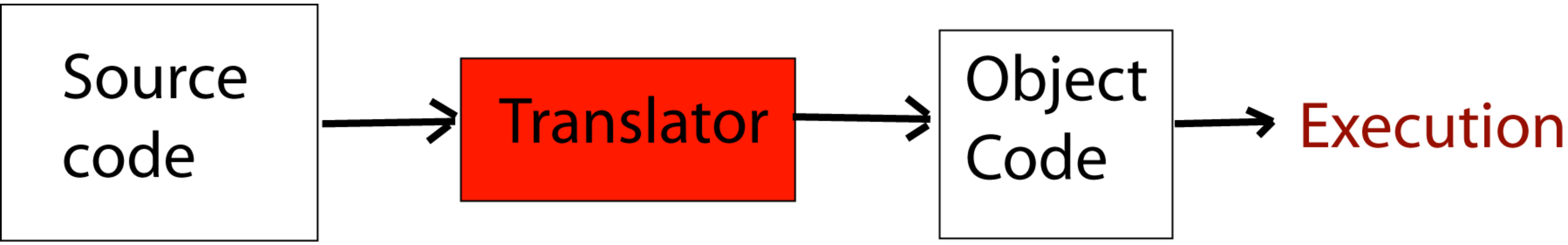
Object code

- **Object code is a program code in machine language that is ready for execution by the computer.**

Language Translators (language processors)

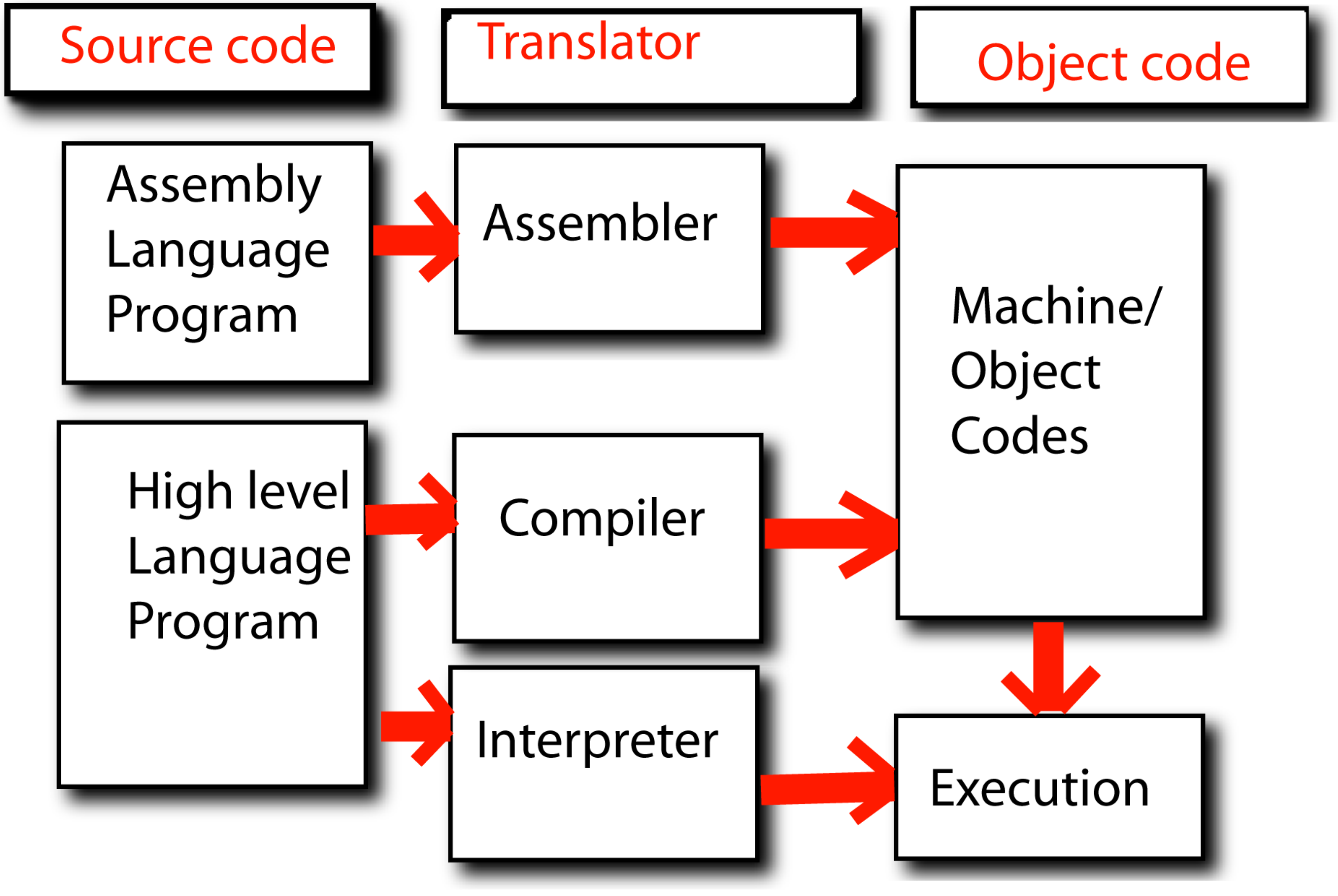
- Language translators are system programs that convert assembly language and high-level language to machine language for the program to execute because Computers work in machine code only.

Programs must be translated into machine/Object codes before execution



There are three types of translators:

1. **Assemblers**
2. **Compilers**
3. **Interpreters**


Source code
Translator
Object code
Assembly Language Program
Assembler
Machine/ Object Codes
High level Language Program
Compiler
Interpreter
Execution

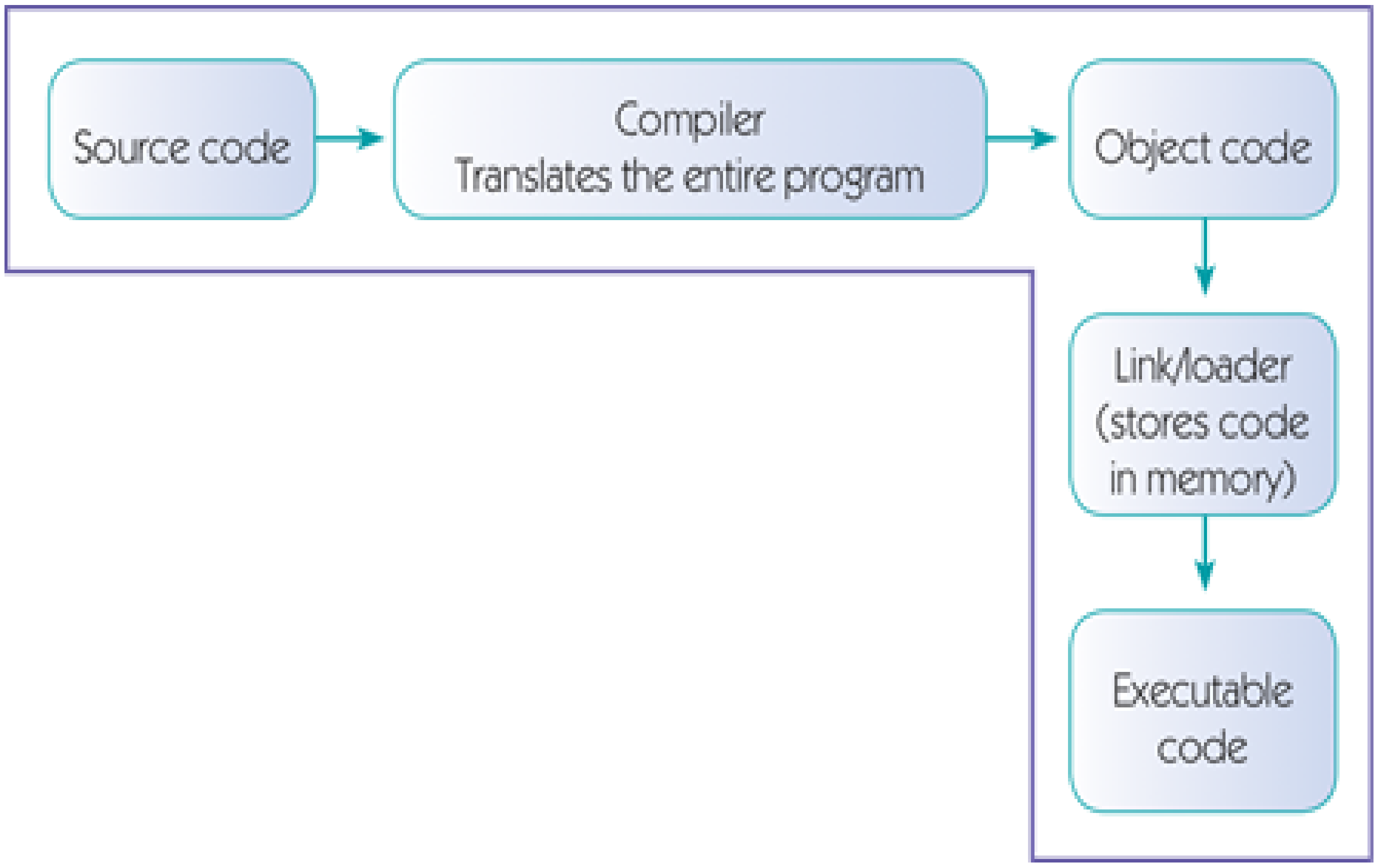
Assemblers

- The assembler is a language that translates an assembly-language program into machine code.

Compilers

A compiler is a computer program that transforms the entire source code written in a high level programming language into object code for execution.

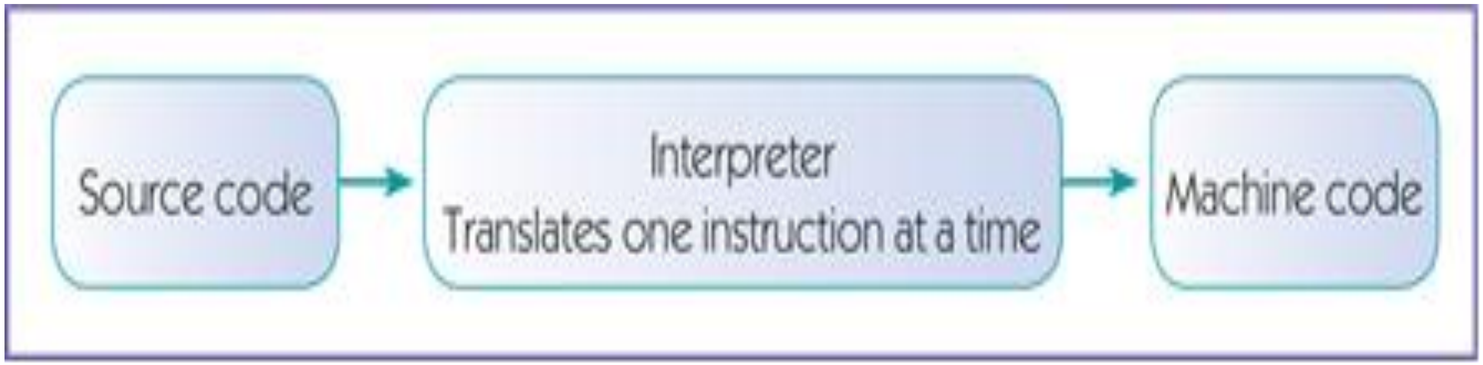
How a Compiler works


Source code
Compiler
Translates the entire program
Object code
Link/loader (stores code in memory)
Executable code

Interpreters

An interpreter is a computer program that directly executes a high level programming language one line at a time to run each time the source code is run because no object code is generated.

How an Interpreter works



Example of a program code (small basic language)

```
TextWindow.Write("enter the Temperature in Fahrenheit ")
fahr = TextWindow.ReadNumber()
celsius = (5 * (fahr - 32) / 9)
TextWindow.WriteLine("This Temperature in celcius is " + celsius +" degrees")
```

```
fahr = input("Please enter the current Temperature in fahrenheit:")
Celsius = (5 * (fahr - 32)//9)
print  ("The temperature in Celsius is“, Celsius, "degree")
```

- TextWindow.title= “Area of Triangle”
- TextWindow.writeline(”What is the length of the triangle?”)
- length= TextWindow.read()
- TextWindow.writeline(”what is the height of the triangle?”)
- height= TextWindow.Read()
- area= 1/2*length*height
- TextWindow.writeline(”the area of the triangle is ” + area + ” cm2”)

```
length = input ("What is the area of the
Triangle?:")
height = input ("What is the Height of the
Triangle?:")
area = (1//2 * length)* height
print ("The area of the triangle is“, area, "squire
cms")
```

```
TextWindow.title="My first program"
TextWindow.write("What is your name?")
Username= TextWindow.Read()
TextWindow.Writeline("What is your year of
birth? "+ username)
yearofbirth = TextWindow.ReadNumber()
TextWindow.Write("enter current year ")
currentyear = TextWindow.ReadNumber()
age = currentyear - yearofbirth
TextWindow.WriteLine("You are " + age + "
years old " +  Username)
```

```
TextWindow.Write("Enter the temperature today (in F):")
temp = TextWindow.ReadNumber()
 If temp > 100 Then
TextWindow.WriteLine("It is pretty hot.")
 ElseIf temp > 70 Then
TextWindow.WriteLine("It is pretty nice.")
ElseIf temp > 50 Then
TextWindow.WriteLine("Don't forget your coat.")
 Else
TextWindow.WriteLine("Stay home.")
EndIf
```

```
height = input ("enter height:")
base = input ("enter base:")
area = 1//2 * (base)*(height)
print ("area of the triangle is" +area)
```

Object-Oriented programming (OOP)

- **Object-oriented programming is a programming language model organized around objects and data.**

Objects

- an object is a self-contained component that has properties and methods needed to make a certain type of data useful.
- An object’s properties are what it knows and its methods are what it can do.
- An object can be a variable, function, or data structure

Control of program flow

- **Program flow is controlled by using Operators.**

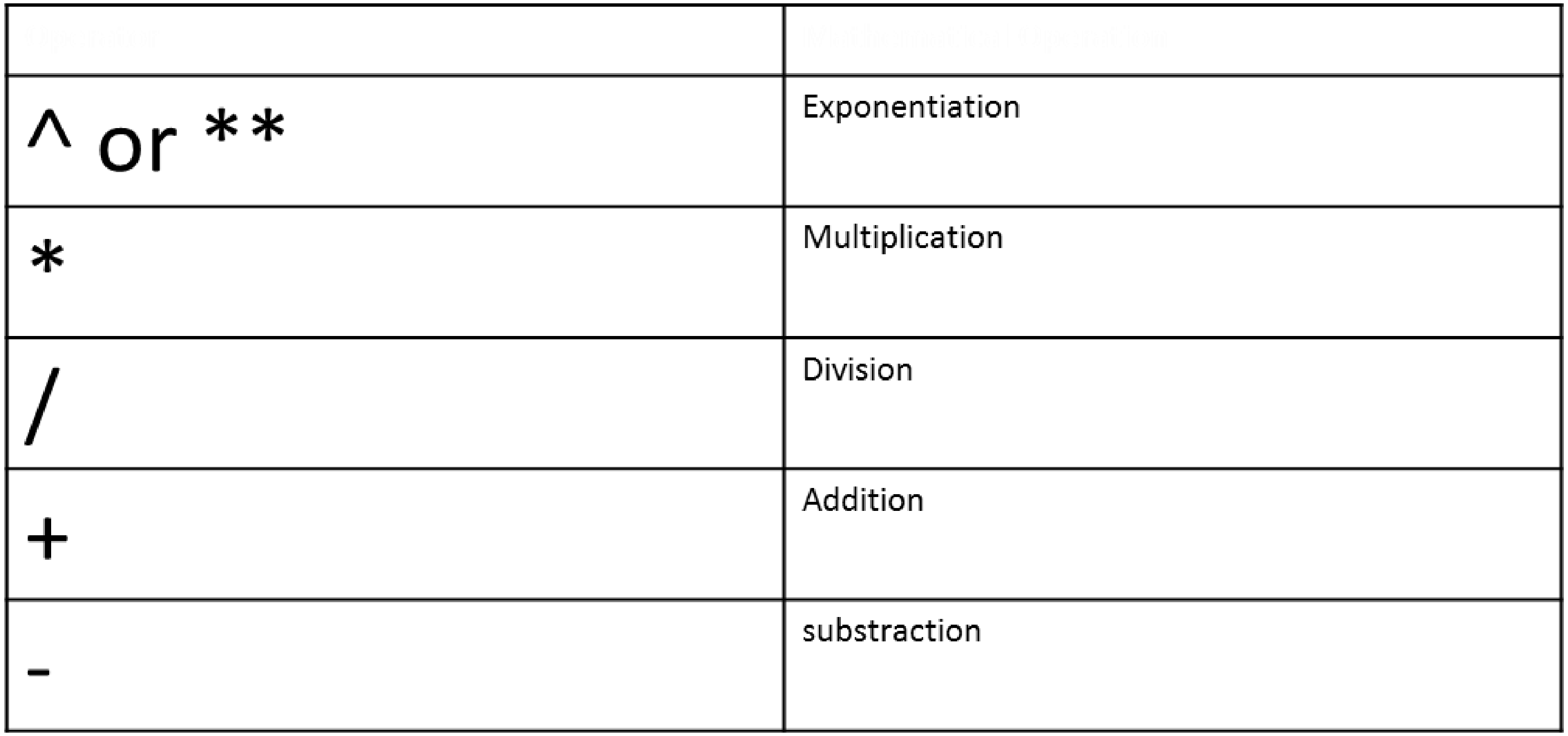
Operations/operators

- These are symbols used to do mathematical or logical procedures. Common simple examples include arithmetic, conditional/comparison, and Boolean operators

Arithmetic operators

These are operators used to carry out arithmetic operations

Arithmetic/numeric operators



Operator	Mathematical Operation
^ or **	Exponentiation
*	Multiplication
/	Division
+	Addition
-	substraction

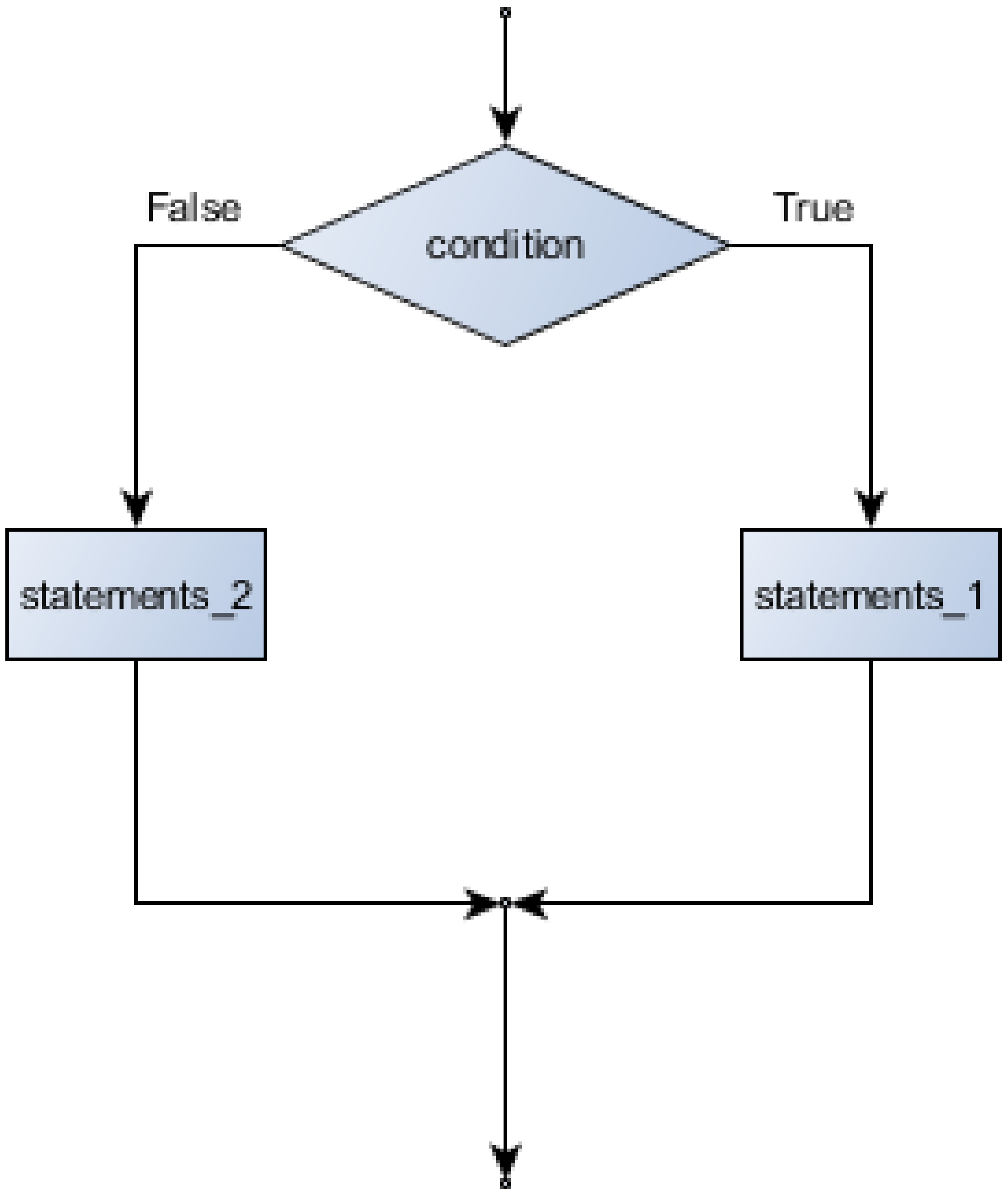
Operands

In mathematics, an operand is a quantity upon which a mathematical operation is performed.

```
height = input ("enter height:")
base = input ("enter base:")
area = 1//2 * (base)*(height)
print ("area of the triangle is" +area)
```

Comparison/conditional/relational operators

These are used to compare two or more expressions or variables to see whether they are equal or one value is greater or less than the other value, then the program decides what actions to take, e.g. whether to execute a program or terminate the program. Or returning a Boolean expression of True or false


False
condition
True
statements_2
statements_1

Comparison Operators

Operator	Meaning
=	Equal to
>	More than
<	Less Than
>=	More than and equal
<=	Less than and equal
<>	Not Equal to

```
if (BOOLEAN EXPRESSION):
    STATEMENTS_1        # Executed if condition evaluates to True
else:
    STATEMENTS_2    # Executed if condition evaluates to False
```

```
if (BOOLEAN EXPRESSION):
    STATEMENTS_1        # Executed if condition evaluates to True
else:
    STATEMENTS_2   # Executed if condition evaluates to False
```

```
score = input ("enter student mark")
if score > 50:
        print ("Pass")
else:
        print ("failure")
```

Boolean/logic operators

When selection is based upon one or more expressions/decisions being true or false, it is possible to combine the expressions/decisions together using the Boolean operators *AND* or *OR* or *NOT*

□If the AND operator is used, *both* conditions must be met in order for the *total* expression to be true.

□If the OR operator is used, *either* condition must be met in order for the *total* expression to be true.

- For the NOT operator, the statement is true if it is contrary to the condition.

With the AND operator

- Both conditions/operands must be met for the expression to be true; if one is false, then condition is false.

Example

- A club plays football only on Sundays and only if it is not raining. Read the day and the weather and print ‘Game on’ if it is a suitable day and weather for playing.

Truth table that describes the semantics of ***and*** (all possible combinations of a and b)

a	**b**	**a *AND* b**
False	False	False
False	True	False
True	False	False
True	True	True

With the OR operator

if either condition is true then the action is taken (true). So, if the day is ‘Sunday’, regardless of the weather, the game is on. If the weather is ‘No Rain’, regardless of the day, the game is on:

The truth table describing o***r***:

a	**b**	**a *or* b**
F	F	F
F	T	T
T	F	T
T	T	T

Xor

exclusive ***or*** is a logical operator that outputs true only when inputs differ (one is true, the other is false)

A	B	A xor B
0	0	0
1	0	1
0	1	1
1	1	0

logical operator, ***not***, only takes a single operand, so its truth table only has two rows:

a	**not a**
F	T
T	F

Key (reserved)words

- Keywords are reserved words with a special meaning for a particular programming language, therefore cannot be used as names for programming elements such as variables and procedures.
- Examples of common reserved words include: do, for, if, while, auto, case, and many others.

Naming and storing data

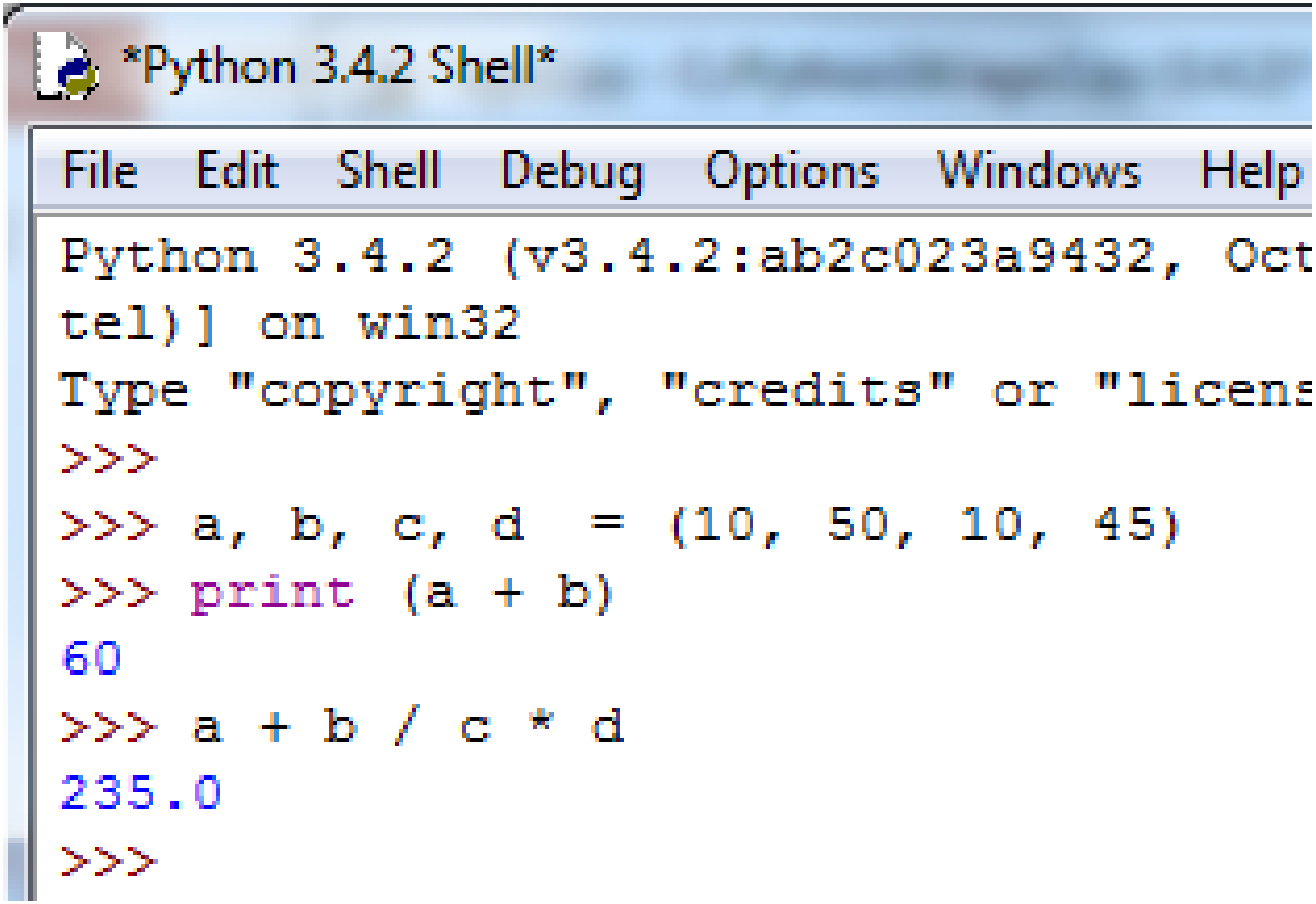
- The instructions in the program must have data to work with.
- e.g. To add two numbers together requires an add instruction.
- The programmer must give the two numbers names for the computer to identify the items.
- These must be stored in computer memory for the computer to use the names to identify the data.

- Data is stored either as constants or as Variables
- Constants have values that are fixed for the duration of the program.
- For example; in case of a program to convert miles to Kms a conversion rate constant can be set as
- Convert_miles_to_Km = 1.6
- (ie there about 1.6 Kms to the mile)
- Pi = 22/7

Variables

A variable is a storage location in computer memory for data in a program. They are a way of naming information for later usage. Each variable has a name whose values can change. The equal (=) sign is used to assign a value to a variable.

```
a = 9
b = 12
c = 3
x = a - b / 3 + c * 2 - 1
y = a - b / (3 + c) * (2 - 1)
z = a - (b / (3 + c) * 2) - 1
print("X = ", x)
print("Y = ", y)
print("Z = ", z
```



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct
tel)] on win32
Type "copyright", "credits" or "licens
>>>
>>> a, b, c, d  = (10, 50, 10, 45)
>>> print (a + b)
60
>>> a + b / c * d
235.0
>>>
```

Examples of acceptable variable names

- Starting Time
- Interest_Value
- TimeOfDay
- Number_of_days
- Name
- Area
- Perimeter
- X
- Y etc.

Rules for naming variables

- No more than 40 characters
- They may include letters, numbers, and underscore
- The first character must be a letter
- You must not use a reserved word
- The equal sign (=) is used to a sign a value to a variable
- E.g. Width = 10
- No parenthesis are used to define the value of a variable
- They are case sensitive

Data types used as Variables

- Numeric/numbers , strings, Boolean, and array.
- Numeric data is either integer or decimal numbers. Decimal numbers are floats/ floating points. Numeric data can be positive or negative.

 E.g. counter = 100 this is *an integer variable.*

- A decimal is a number with a decimal point. E.g. Miles = 1000.01 this is *a floating point variable.*

- String. a string is a set of characters in between quotation marks that can be a name, a string of numbers, a sentence, a paragraph etc.

e.g. name = "Peter" is a string variable

The plus (+) sign is the string concatenation operator.

e.g. print "hello world"+ "its 10 PM"

String concatenation is the operation of joining character strings.

```
 S = “Hello World”
Print (s)
Hello World
```

Note that a variable can be redefined. E.g.

Score = 8

Score = score + 1

1. How do you tell Python that a variable is a string (characters) instead of a number?
2 Once you have created a variable, can you change the value that is assigned to it?
3 With variable names, is TEACHER the same as TEACHEr?
4 Is 'Blah' the same as "Blah" to Python?
5 Is '4' the same as 4 to Python?
6 Which of the following is not a correct variable name? Why?
a) Teacher2
b) 2Teacher
c) teacher_25
d) TeaCher

Is "10" a number or a string?

1 Make a variable and assign a number to it (any number you like). Then display your variable using print.

2 Modify your variable, either by replacing the old value with a new value, or by adding something to the old value. Display the new value using print.

3 Make another variable and assign a string (some text) to it. Then display it using print.

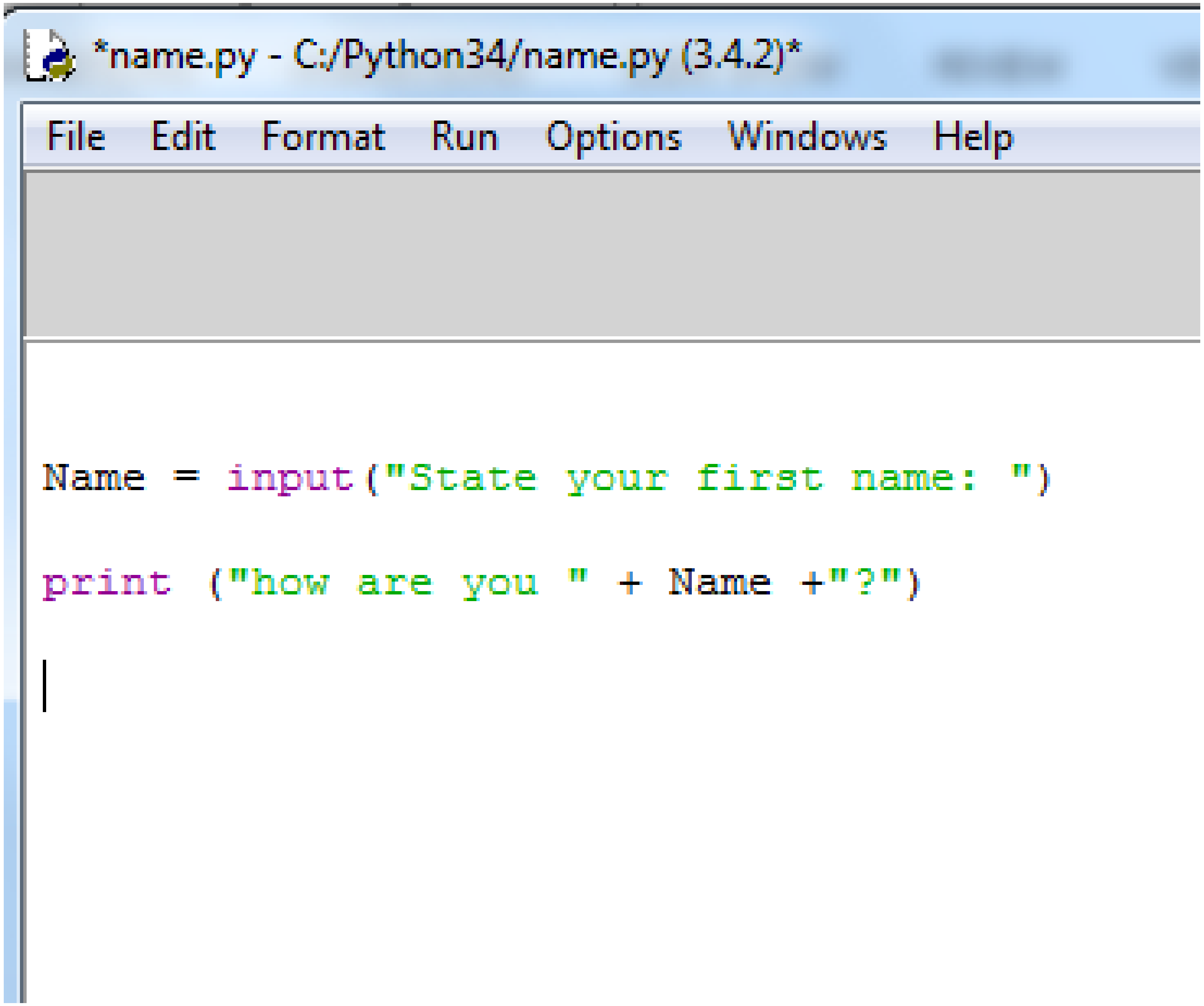
4 calculate the number of minutes in a week. But this time, use variables. Make a variable for DaysPerWeek, HoursPerDay, and MinutesPerHour (or make up your own names), and then multiply them together.

- 5 **How many minutes would there be in a week if there were 26 hours in a day? (Hint: Change the HoursPerDay variable.)**

```
TextWindow.Write("Enter temperature in
Fahrenheit: ")
fahr = TextWindow.ReadNumber()
Celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperature in
Celsius is " + Celsius)
```

```
fahr = ("Enter temperature in Fahrenheit: ")
celcius = 5 * (fahr - 32) // 9
print ("the temperature in celcius is" + celcius)
```

- Strings are always enclosed in quotes. Examples of strings: ("I am a Small Basic programmer")   ("012345"),("Title Author")
- More than one string can be <u>concatenated</u> (combined into one string) with a + operator
- e.g "your age is "+"36 "+years (leave space between the last word and the closing parenthesis to create space between combined word)


name.py - C:/Python34/name.py (3.4.2)
File Edit Format Run Options Windows Help

```
Name = input("State your first name: ")

print ("how are you " + Name +"?")
```

- Boolean Data. A data type that can only represent two values: true or false.
- e.g. fee.paid = false
- Lists. This is an array used to store a set of elements together. E.g. lst=[1,2,3,4]
- Dictionary. A tool that maps individual keys to specific variables. Each key in the dictionary is unique and is mapped to a specific value

Local and Global Variables

- **A local variable is one for only a section of the program**
- **A global variable is one that is available for the entire program.**

Changing Data types

some functions that convert data from one type to another:

`float()` will create a new float (decimal number) from a string or integer.

`int()` will create a new integer from a string or float.

`str()` will create a new string from a number (or any other type).

1 Use float() to create a number from a string like '12.34'. Make sure the result is really a number!

2 Try using int() to create an integer from a decimal number like 56.78. Did the answer get rounded up or down?

3 Try using int() to create an integer from a string. Make sure the result is really an integer!

Using variables: example (small Basic)

```
□number1 = 10
□number2 = 20
□number3 = number1 * number2
□TextWindow.WriteLine(number3)
```

```
TextWindow.title="The age calculator"
currentDate=clock.Year
TextWindow.WriteLine("State your First name")
FirstName=textwindow.Read()
TextWindow.WriteLine("Enter your year of birth "
+ FirstName +" Dear")
yearOfBirth=textwindow.Read()
age=currentDate-yearOfBirth
TextWindow.WriteLine("You are " + age + "
years old, " +FirstName)
```

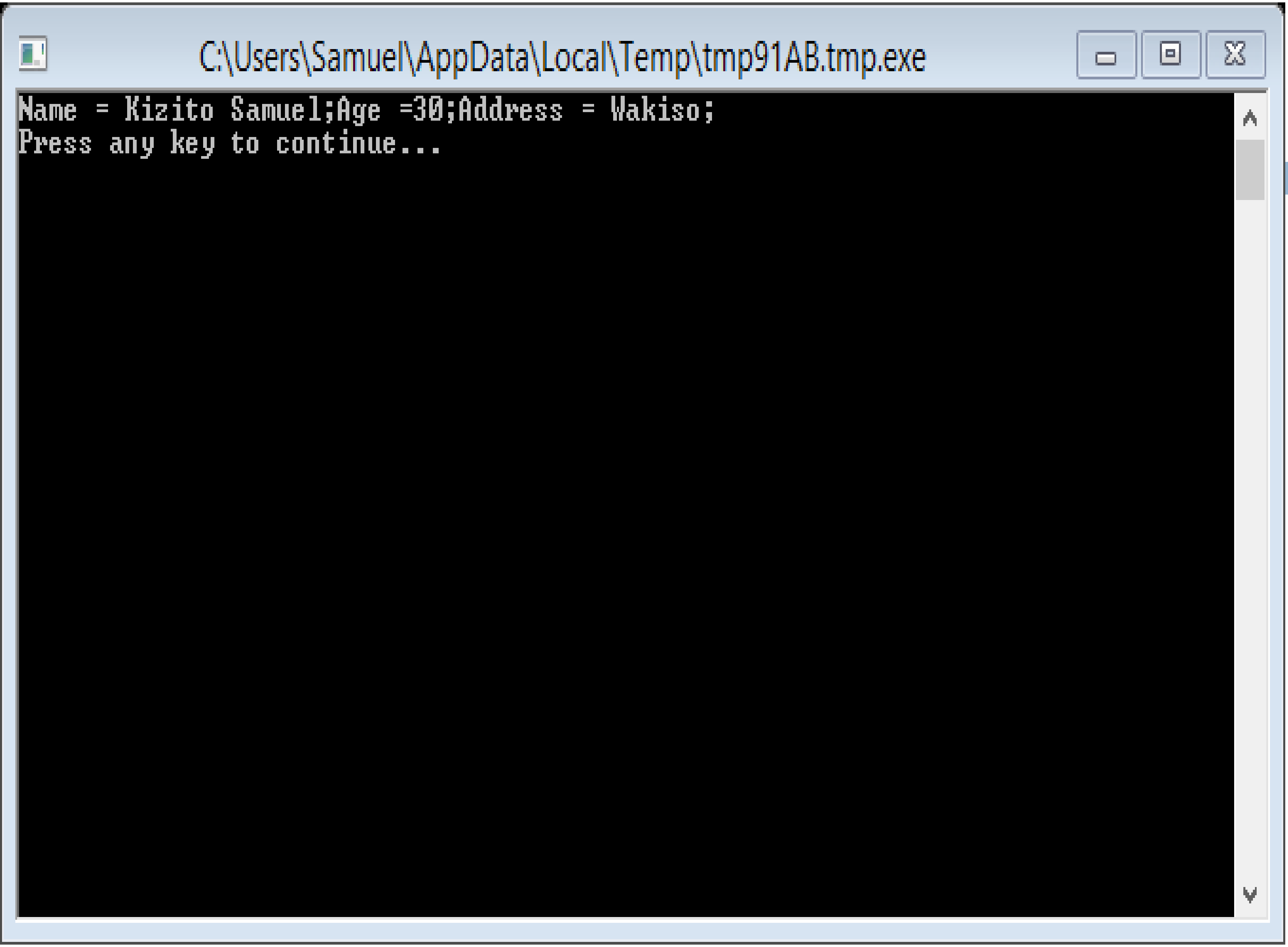
Arrays

- An array is a list or table of data under one variable name/index. each item in the list is an element. For example, the program can store the names of individuals in a class plus their, address and age which can be selected by the index.
- A list can be created by defining it with []

```
list = [2, 4, 7, 9]
list2 = [3, "test", True, 7.4]
```

Example of an array

- person["Name "] = " Kizito Samuel"
- person["Age "] = 30
- person["Address "] = " Wakiso"
- TextWindow.WriteLine(person)



```
Name = Kizito Samuel;Age =30;Address = Wakiso;
Press any key to continue...
```

Procedures/routine

- A sequence of program instruction i.e. a group of instructions that perform a specific task.

Functions

- In programming, a function is a named section of a program that performs a specific task to return a value, i.e. a type of procedure or routine

Types of programming

- Procedural –oriented
- Functional
- logic

Procedural-oriented programming

- Procedural programming uses a set of instructions telling a computer what to do step by step and how to perform from the first code to the next code.
- Procedures, also known as routines, subroutines, methods, or functions contain a series of computational steps to be carried out.

Functional programming

- This is a style of programming which represents computations as the evaluation of mathematical expressions/mathematical functions.
- A mathematical expression is a relation between a set of inputs and a set of outputs e.g. $f(x) = x+3$, that is f of x equals to x plus three.

Logic programming

- Logic programming is a programming pattern based on formal logic/reasoning.
- A set of sentences in logical form, expressing facts and rules about some problem field.

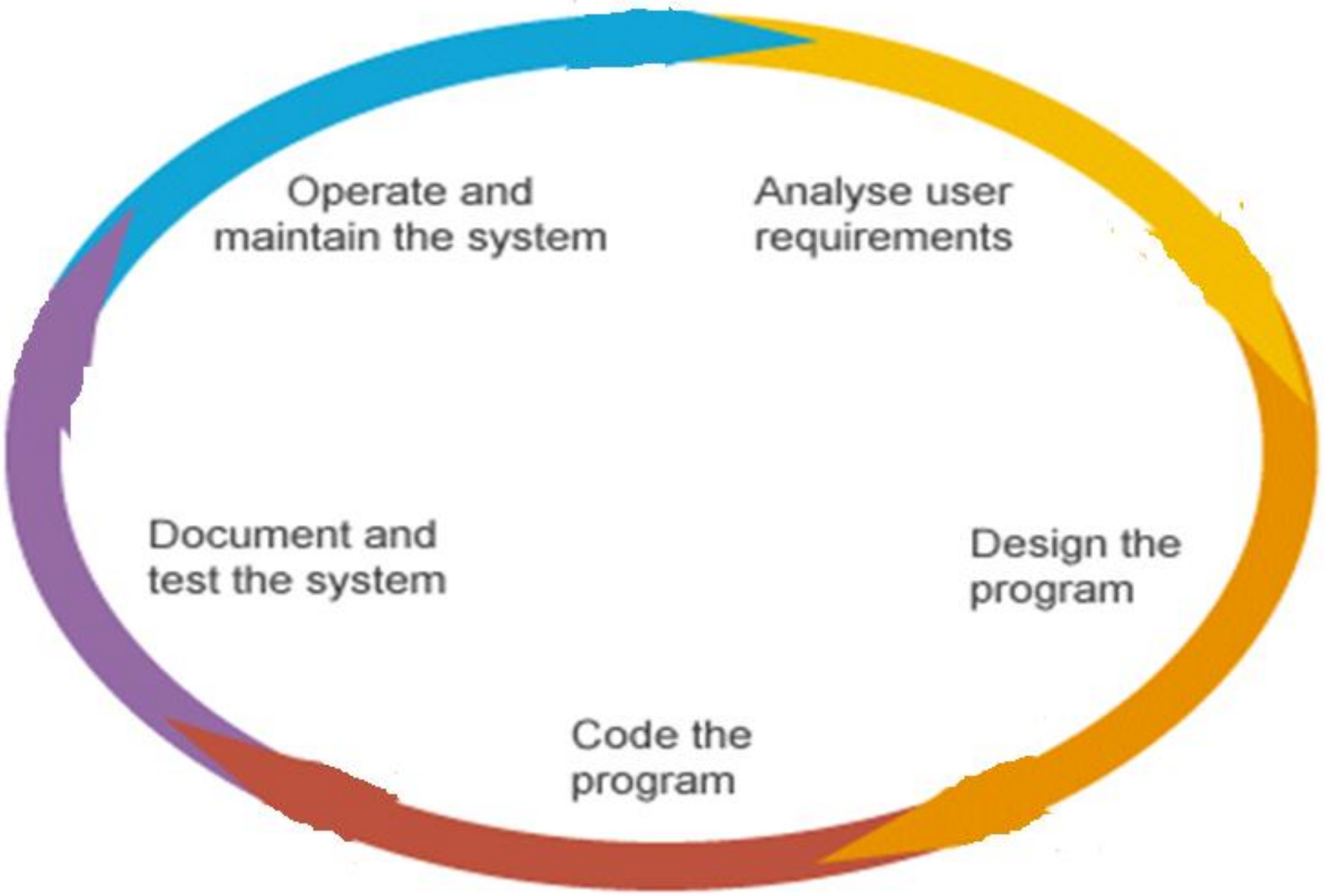
The program development life cycle (PDLC)

PDLC are the stages or steps through which a computer program is created.

PDLC is a continuous process that consists of five/six general steps:

- 1. Analyse problem
- 2. Design program
- 3. Code program
- 4. Test program
- 5. Formalise program
- 6. Maintain program

PDLC


Analyse user requirements
Design the program
Code the program
Document and test the system
Operate and maintain the system

Problem analysis

- This involves identifying the need to overcome a given problem and defining the problem.
for example, the need for a school to process students marks and grade accurately at the end of term.
- Defining the problem involves identifying the user's program objectives, desired inputs, outputs, and processing.

Defining the problem

In defining the problem there must be no ambiguity. The problem should be clear and concise and have only one meaning.
Examples of unambiguous problems are:
i) Calculating the price of an item after a 10% discount
ii) Converting a temperature from ° C to ° F
iii) Computing the average rainfall for the month of May in a certain place.

Analysing the problem

- In order to develop an algorithm (procedure) to accomplish a certain task one must analyse the task as a sequence of instructions that can be performed by the computer.
- These instructions can be divided into three main parts: input and storage instructions, processing instructions, and output instructions.

Program design

- This is the actual development of a program’s processing logic (algorithm).

Programme Algorithm

An algorithm is a sequence of instructions written using special rules and statements which, if followed, produces a solution to the given problem.

Example of an algorithm

Algorithm:
>Read student name and marks obtained.
□Calculate total marks and average marks.
□Write student name, total marks, average marks.

Program design involves three tasks:

□(i) grouping the program activity into modules (a module a logical section in the program)

□(ii) develop a solution algorithm for each module.

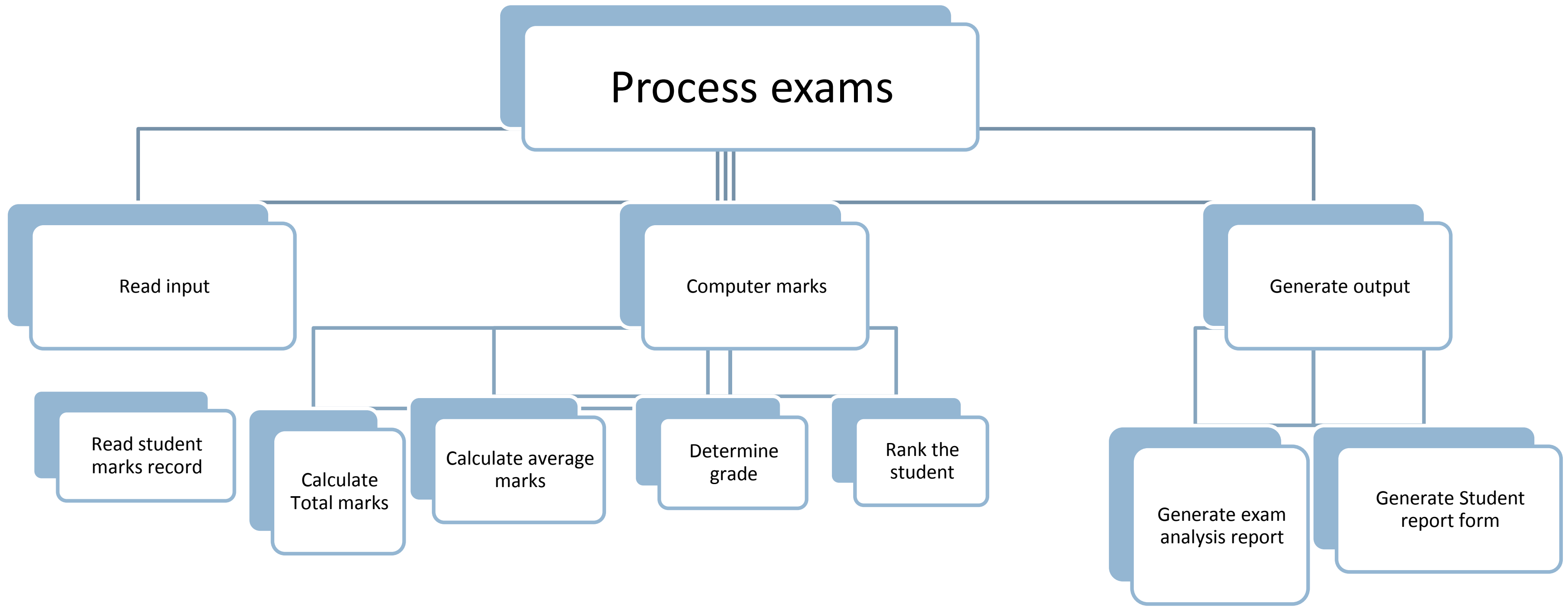
□(iii) test the solution algorithm

Top-down design

- This involves breaking down the programme into smaller, manageable components represented graphically on a hierarchy chart;
- the top most showing the main module referred to as the main routine, which is then subdivided into smaller sections also referred to as sub-routines.
- Start by grouping tasks into modules by focusing on *what* must be done (requirements).

- Each module is represented by a rectangle labeled by its name. e.g. To calculate pay, there may be need to create submodules to compute regular pay, overtime pay, and compute deductions

Top-down chart



the Design details using Pseudo-code, flowcharts and control structures

- The two ways to show program details is either using pseudo-code or drawing the details using flowcharts, or both.
-

Pseudo-code

- A Pseudo-code is a narrative/descriptive form using human language statements to describe the logic and processing flow of a program.

Pseudo code

```
START
READ student name, mark1, mark2, mark3, mark4
Totalmarks = mark1 + mark2 + mark3 + mark4
Averagemark = Totalmarks / 4
PRINT student name, totalmarks, averagemarks
STOP
```

Pseudo-code commonly used keywords

Pseudo-code uses several keywords to indicate common input, output, and processing operations:

- **Input:** **READ, READLINE, OBTAIN, GET, ENTER, INPUT**
- **PROCESSING:** **Initialize, SET, INIT Add one, INCREMENT, BUMP, COMPUTE, CALCULATE, DETERMINE**
- **Output:** **PRINT, DISPLAY, SHOW, WRITE, WRITELINE**

Task

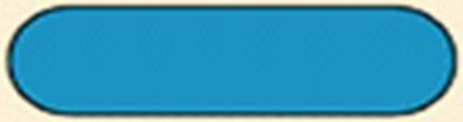
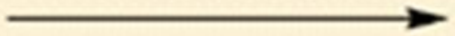
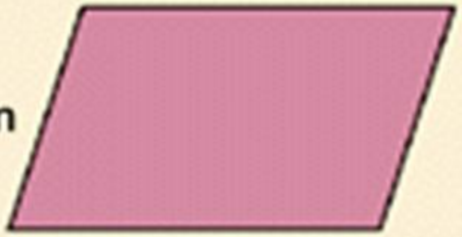
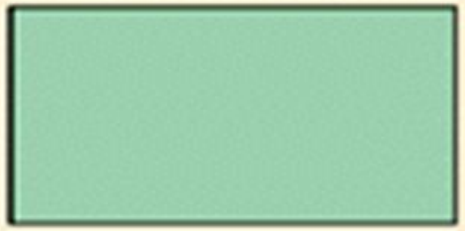
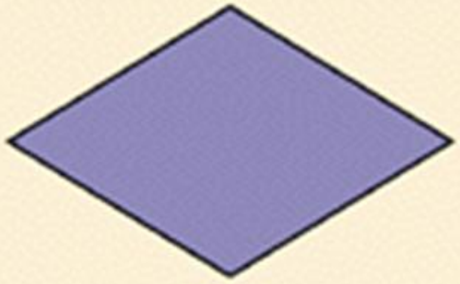
Write out a program pseudo-code for the following problems:

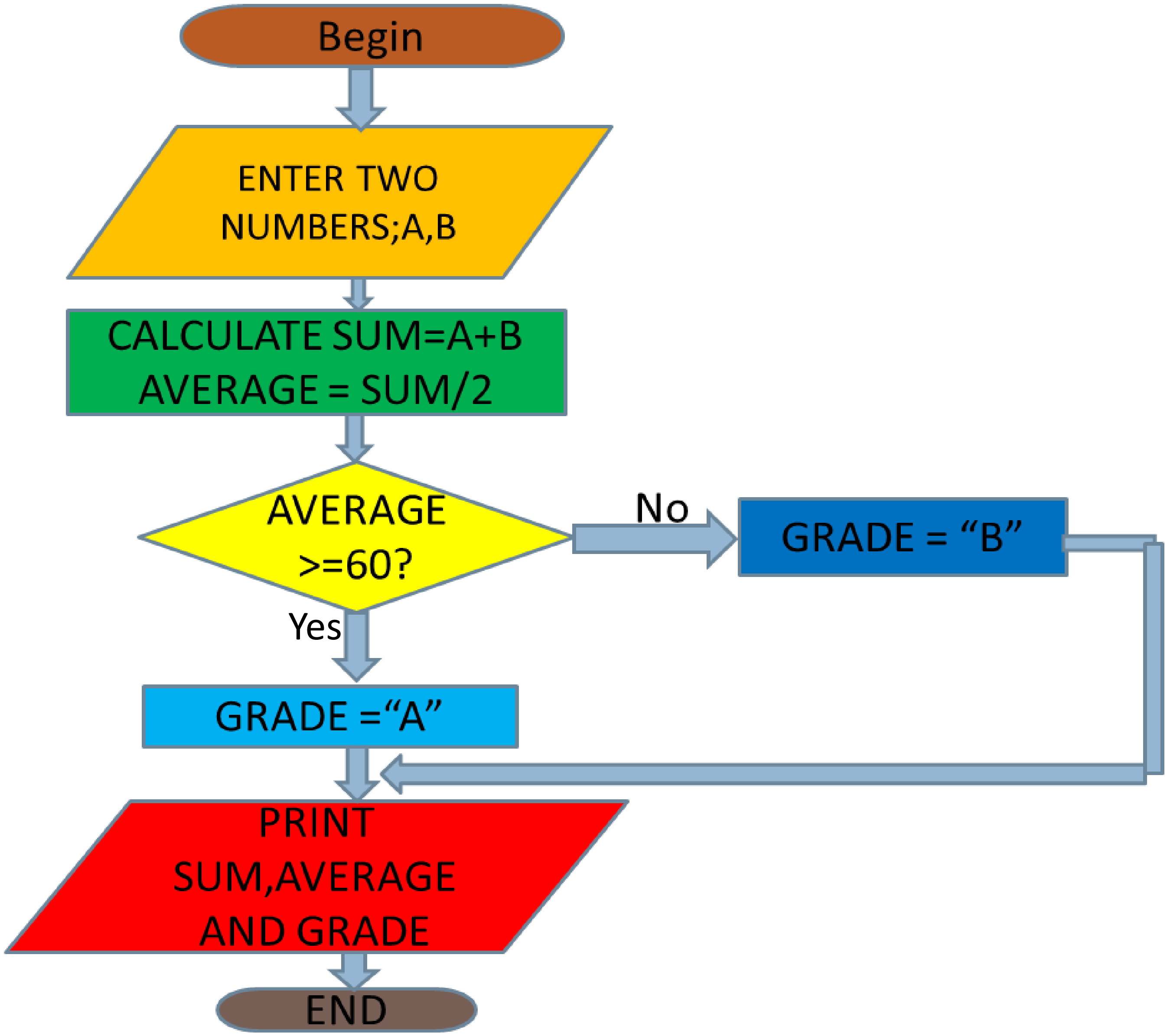
- Converting degrees Celsius to Fahrenheit
- Calculating the age of an individual
- Any other?

Program flowchart

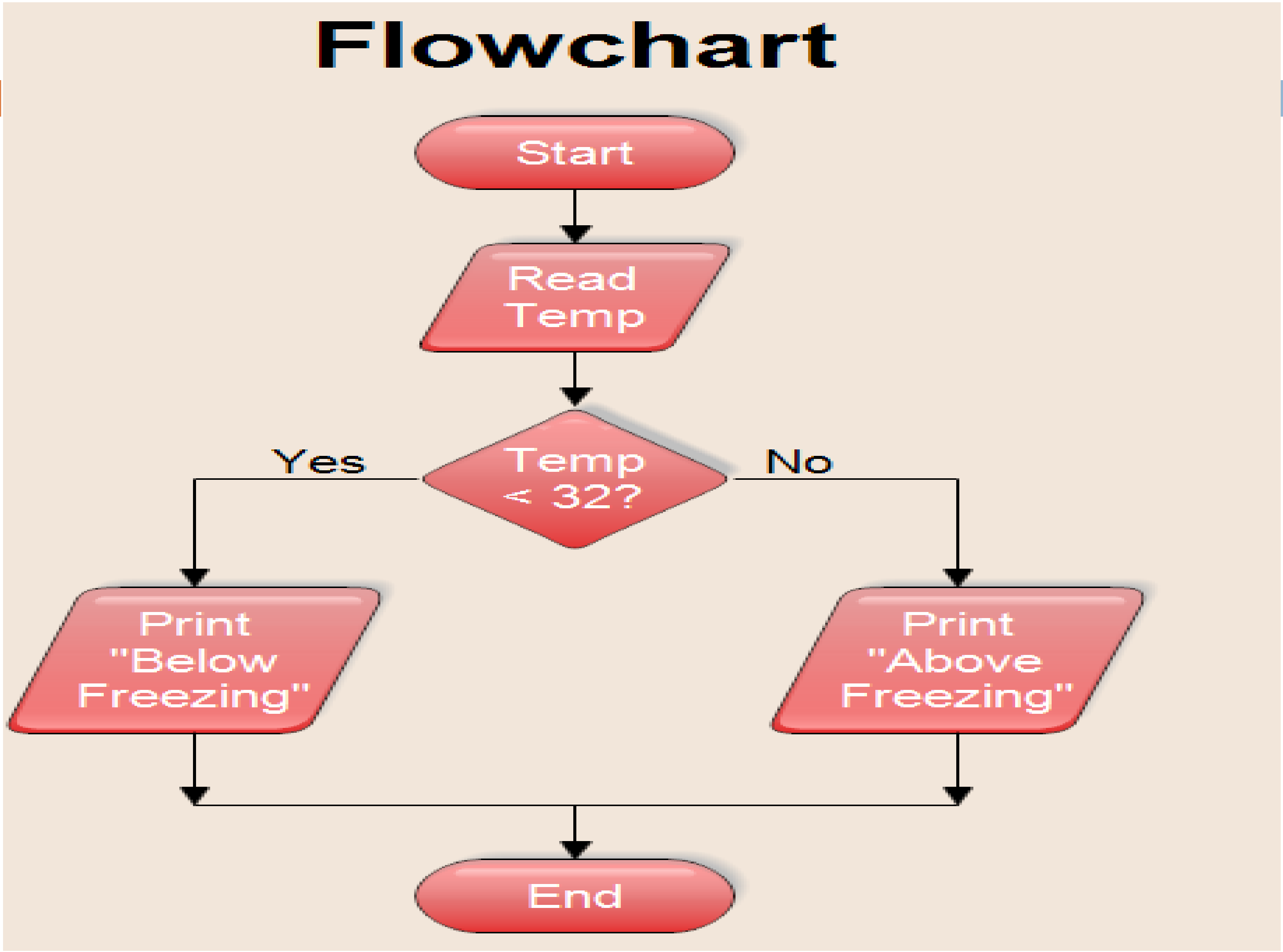
This is a diagrammatic representation of a program's algorithm using standard symbols and short statements to describe various activities.

- These symbols are called ANSI symbols (called after the American National Standards Institute that developed them).
- A flowchart shows how a program works before one begins the actual coding of the program.

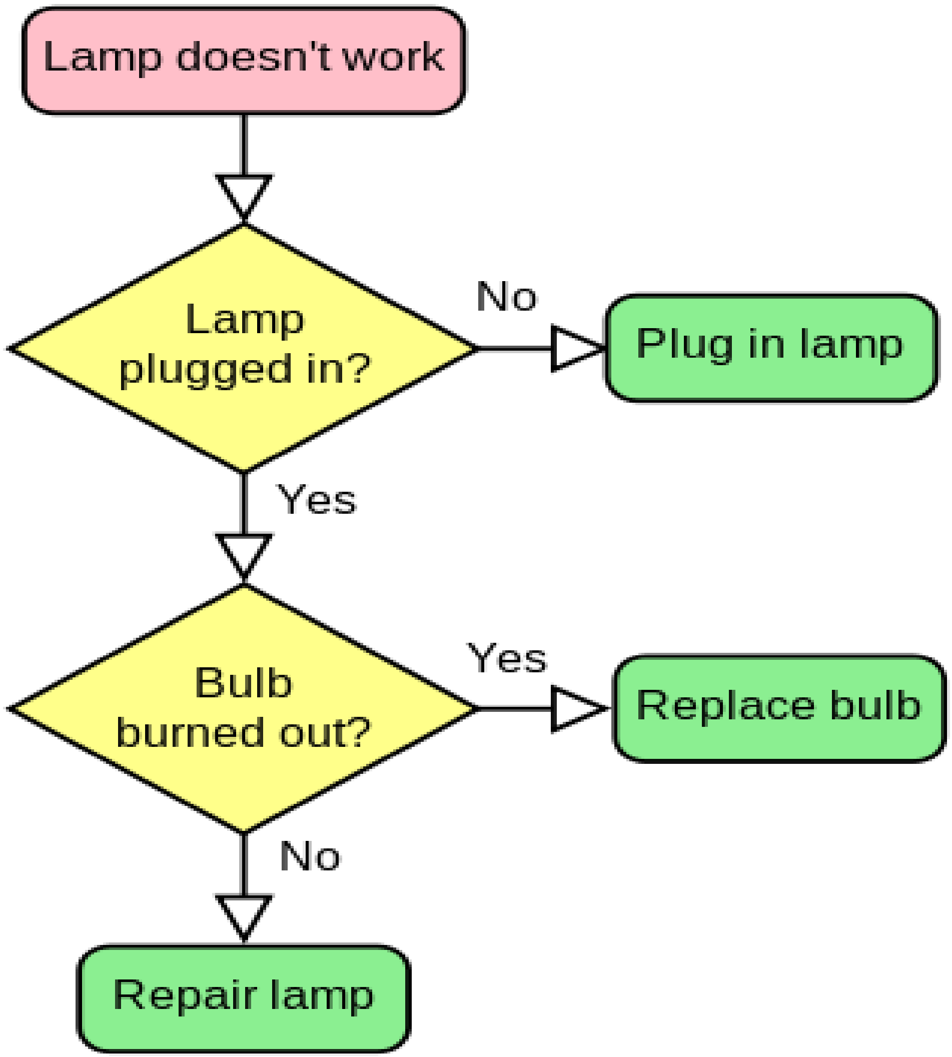
Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of a program.
Flow line		Denotes the direction of logic flow in a program.
Parallelogram		Denotes either an input operation (e.g., INPUT) or an output operation (e.g, PRINT).
Rectangle		Denotes a process to be carried out (e.g., an addition).
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g., IF/THEN/ELSE).


Begin
ENTER TWO NUMBERS;A,B
CALCULATE SUM=A+B
AVERAGE = SUM/2
AVERAGE >=60?
No
GRADE = “B”
Yes
GRADE =“A”
PRINT SUM,AVERAGE AND GRADE
END

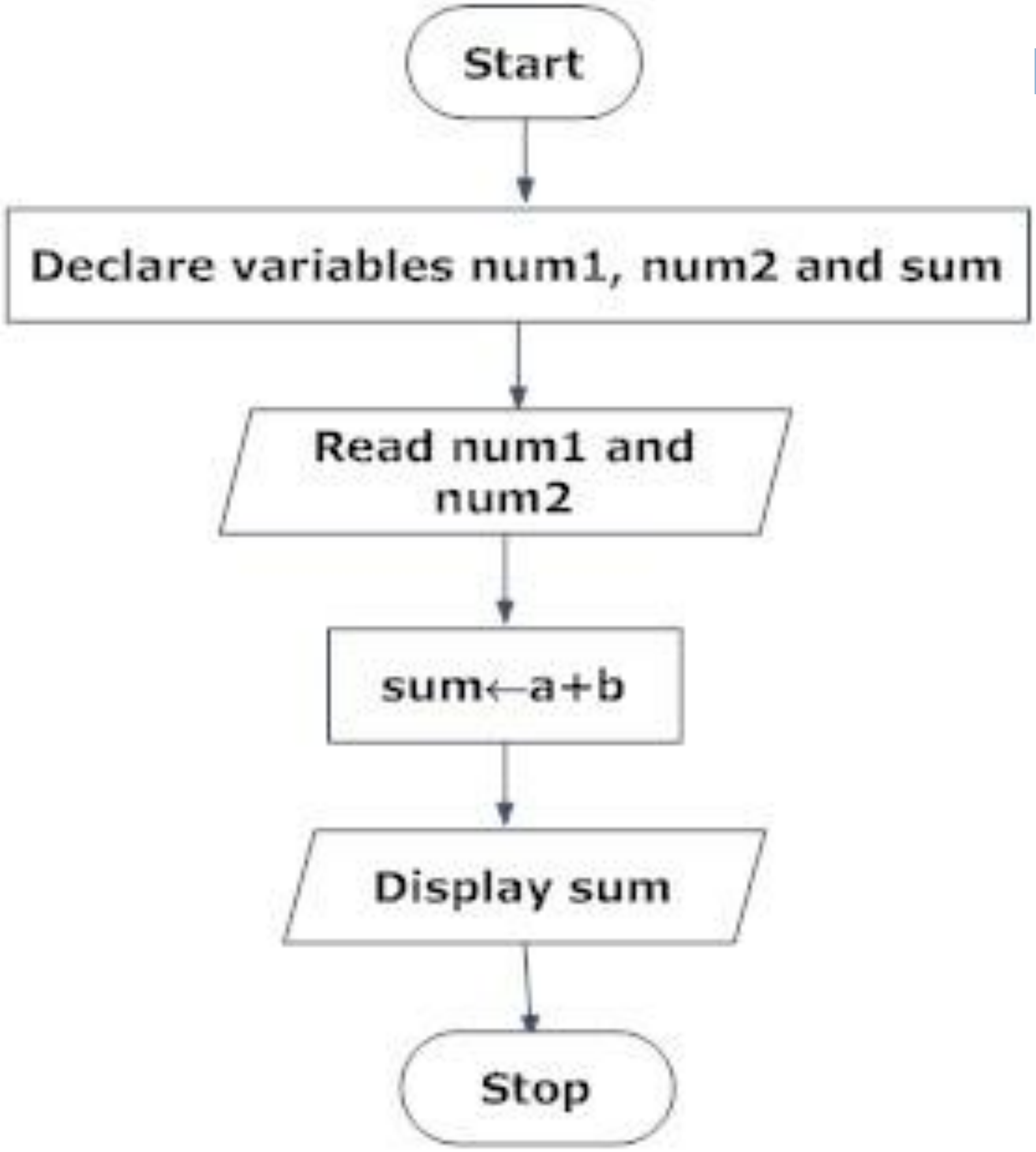
Flowchart (input-output)



Lamp flowchart

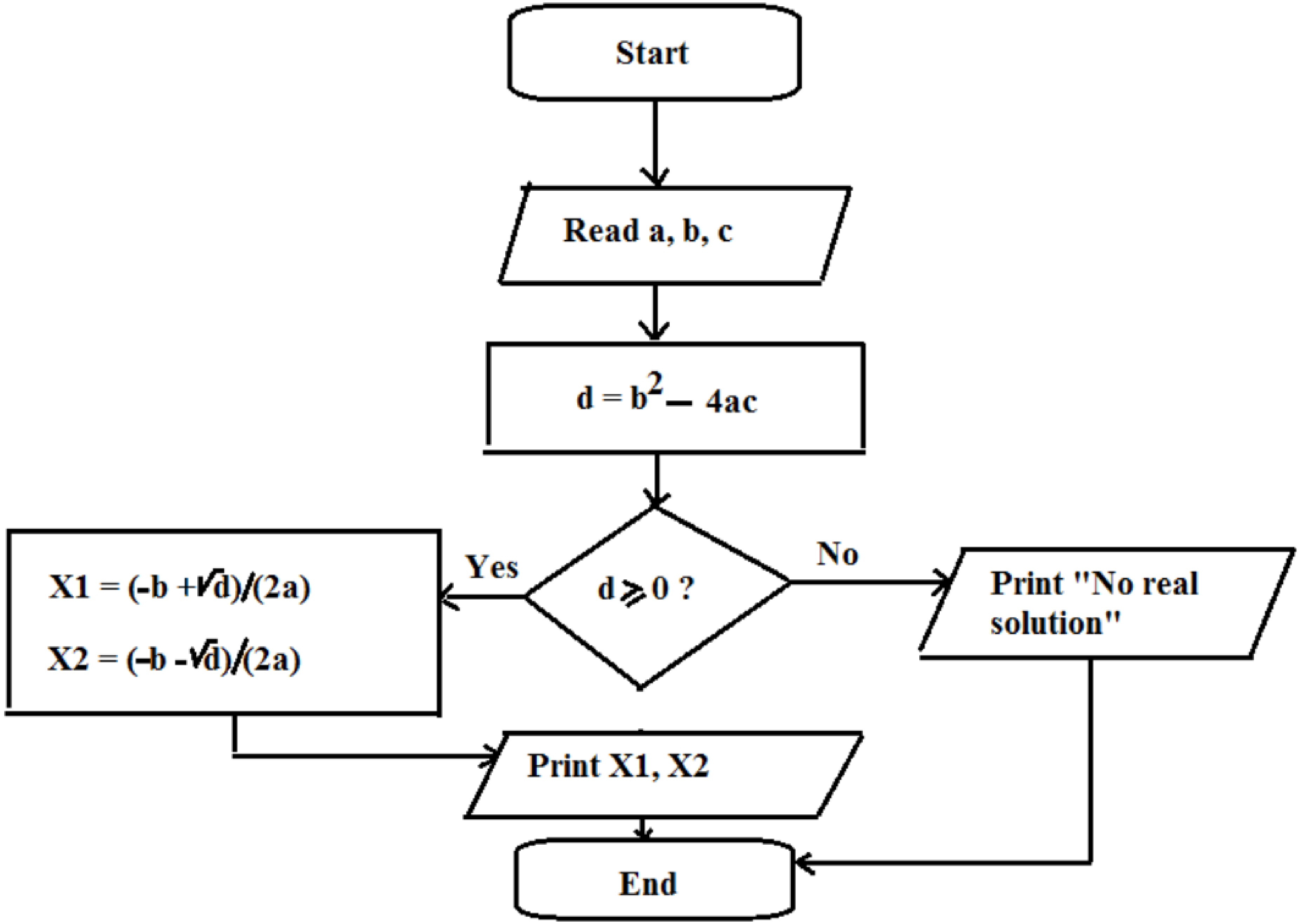


Flowchart to add two numbers entered by user



```
Num_1 = 12
Num_2 = 6
sum = Num_1 + Num_2
print (sum)
```

a flowchart to find all the roots of a quadratic equation $ax^2+bx+c=0$


Start
Read a, b, c
d = b2 − 4ac
d ≥ 0 ?
Yes
No
X1 = (-b +√d)/(2a)
X2 = (−b -√d)/(2a)
Print "No real solution"
Print X1, X2
End

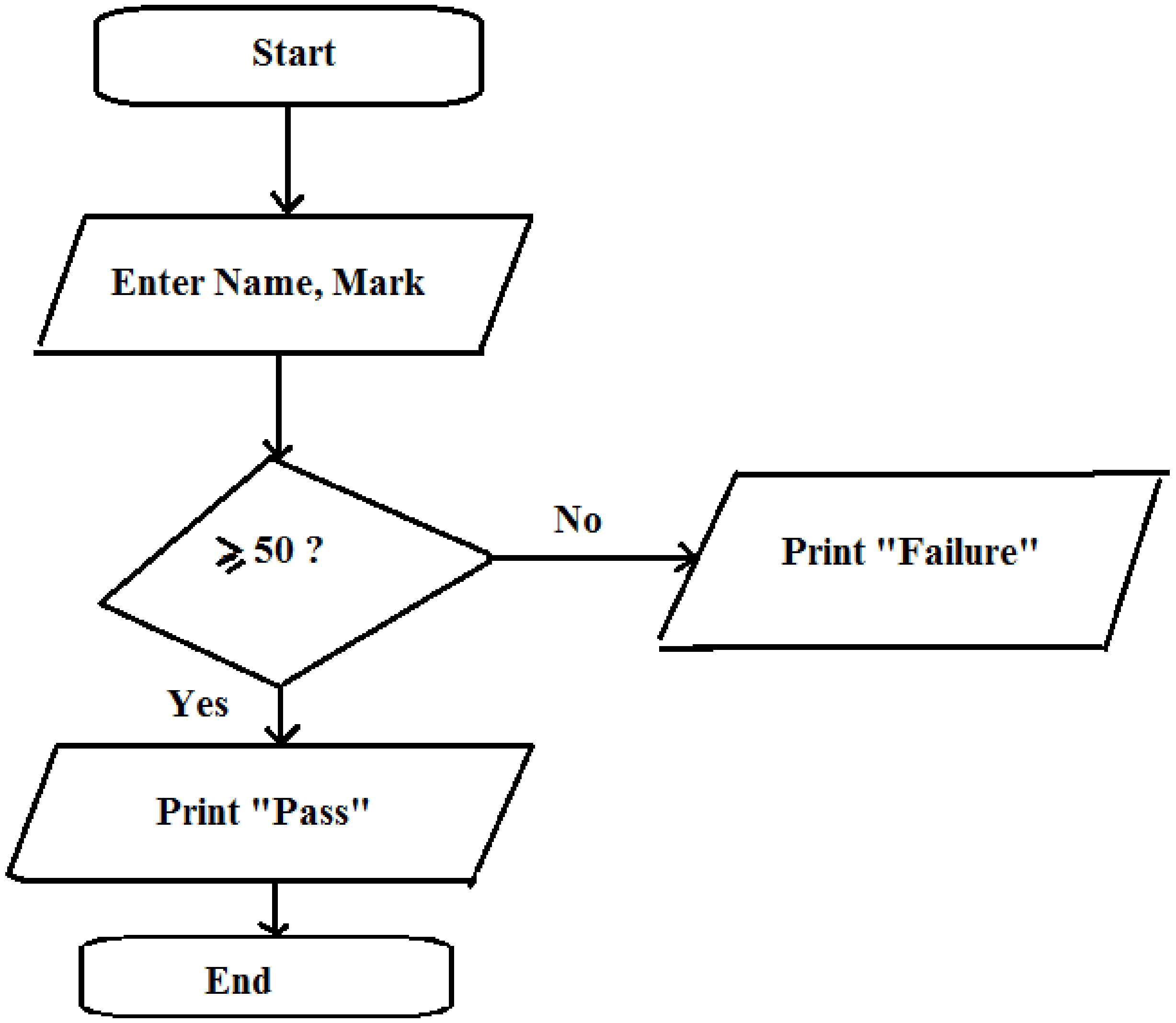
a quadratic equastion is any equation in the form $ax^2 + bx + c = 0$ where $a \neq 0$
any quadratic equation can be can be solved with the formula
$x = (-b \pm \sqrt{(b^2 - 4ac)})\ /2a$
The discriminant, d of the equation is the expression b^2-4ac

Task

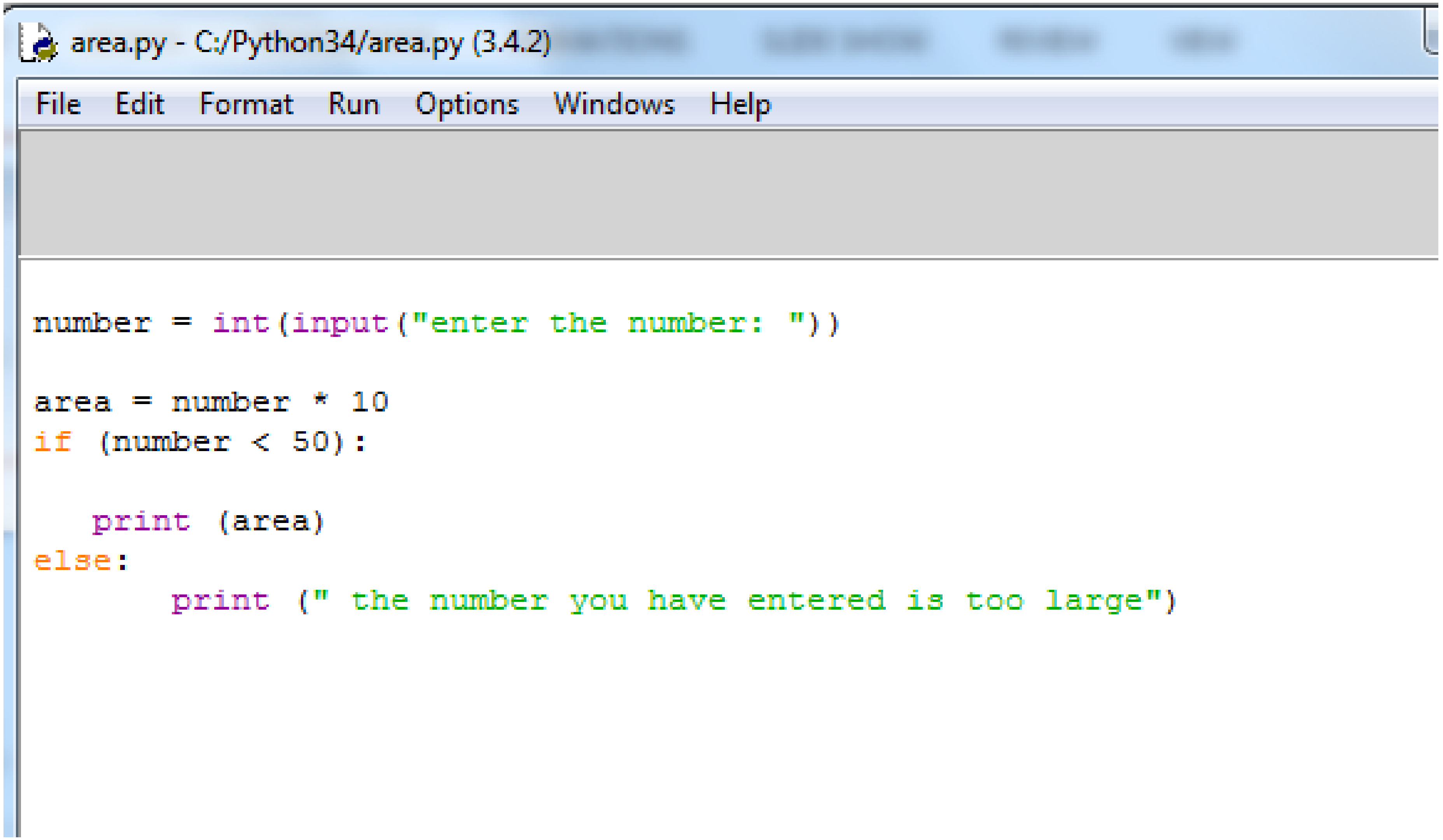
Use a flow chart to map out a program logic (algorithm):

1. Determine performance of candidates, where a score above 50% is a pass while less than 50% is a failure.
2. Write an algorithm to determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks. Passing is 40% and above

Determine performance of candidates, where a score above 50% is a pass while less than 50% is a failure.

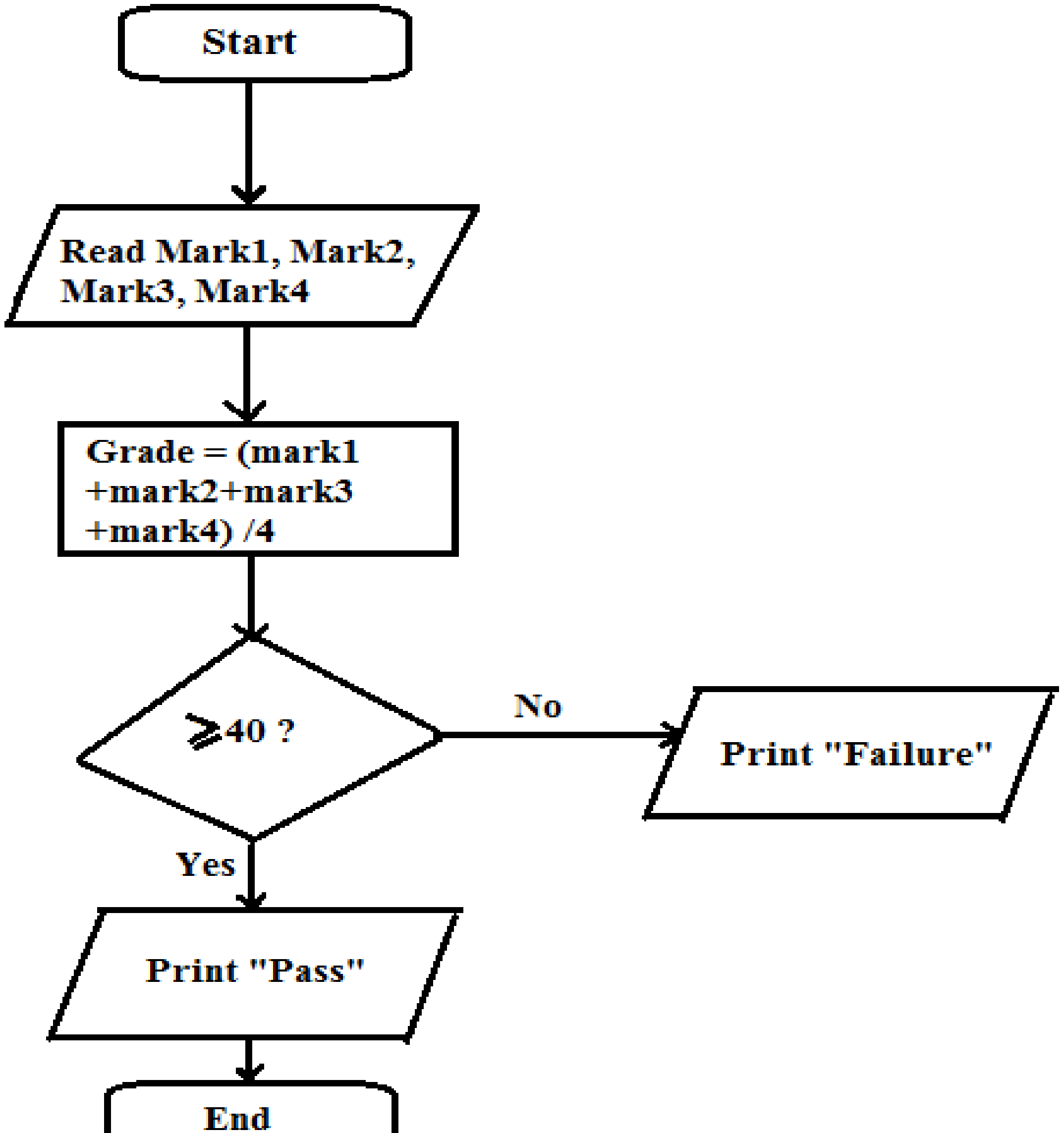


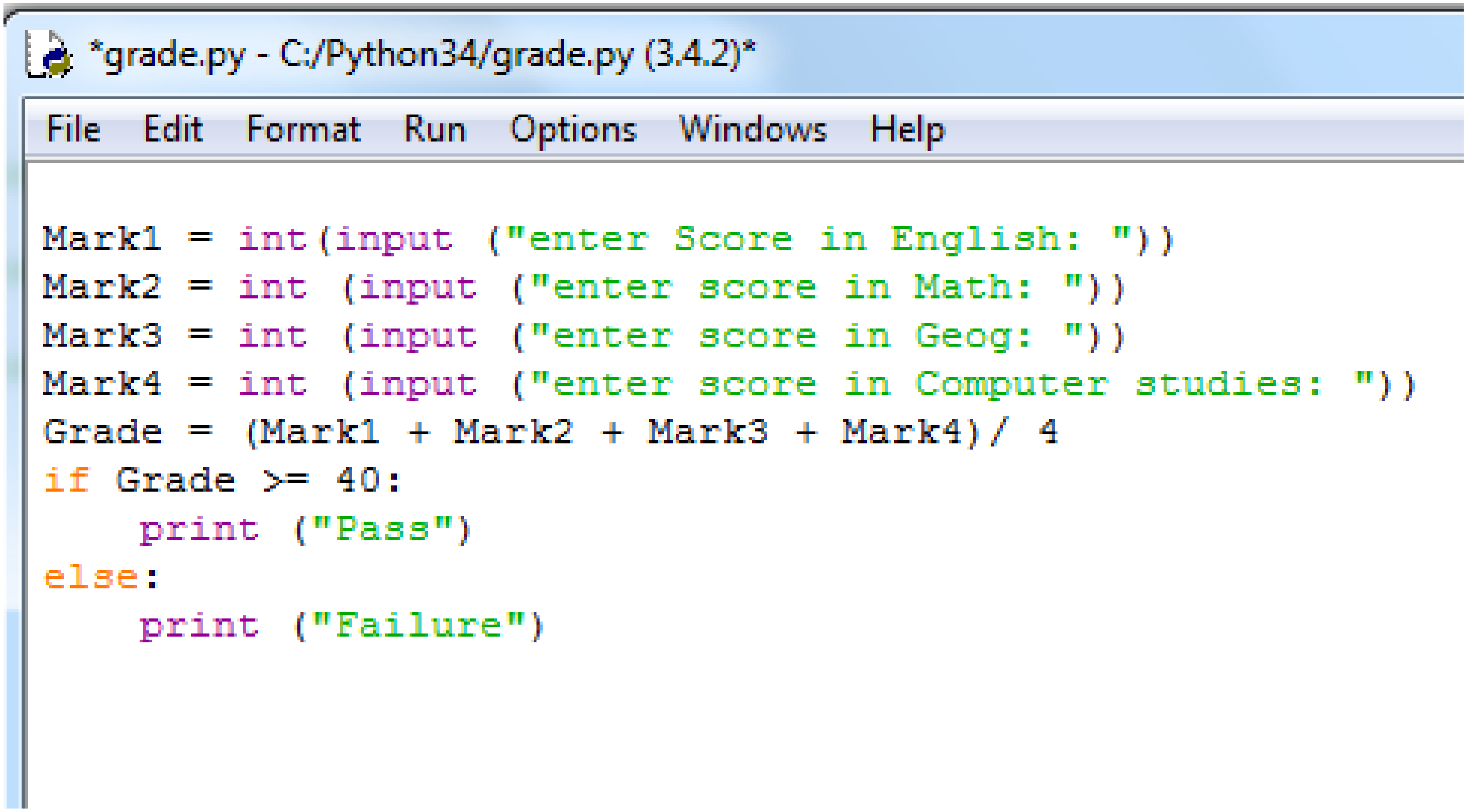
```
score = int (input ("enter student mark: "))
if score > 50:
        print ("Pass")
else:
        print ("failure")
```


area.py - C:/Python34/area.py (3.4.2)
File Edit Format Run Options Windows Help

```
number = int(input("enter the number: "))

area = number * 10
if (number < 50):

   print (area)
else:
       print (" the number you have entered is too large")
```

determine a student's final grade and indicate whether it is passing or failing. The final grade is calculated as the average of four marks. Passing is 40% and above


Start
Read Mark1, Mark2, Mark3, Mark4
Grade = (mark1 +mark2+mark3 +mark4) /4
≥40 ?
No
Print "Failure"
Yes
Print "Pass"
End


grade.py - C:/Python34/grade.py (3.4.2)
File Edit Format Run Options Windows Help

```
Mark1 = int(input ("enter Score in English: "))
Mark2 = int (input ("enter score in Math: "))
Mark3 = int (input ("enter score in Geog: "))
Mark4 = int (input ("enter score in Computer studies: "))
Grade = (Mark1 + Mark2 + Mark3 + Mark4)/ 4
if Grade >= 40:
    print ("Pass")
else:
    print ("Failure")
```

Coding the program

- This is the actual writing of the program to get a program source code using a specific programming language once the design has been developed and reviewed.
- Computer programs are written/coded using the specific rules and statements of the particular computer programming language being used.

Save the program and run it. If the program doesn’t do what you expect, or you get any error messages, try to fix it and make it work.

Factors to consider in Selecting the program to use

- Ease of learning the language. The easier the quicker for one to learn and use it to come up with a program. Ease of understanding makes it easy to use and debug.
- Speed of development of a program using the language. That is how long it can take you to code or debug using a particular language

- Portability of the language. That has a standardised language syntax and platform environment
- Fit for the purpose. Some languages are good for specific types of programs and not others. e.g. Some are good for designing games while others for system programs, or business applications
- The level of expertise/skill of the programmer. Some languages are good for beginners and not others

- **The ability of the program to interact with other existing programs**
- **Availability of help facility to ensure correct coding. e.g. Type checking facility to minimise syntax errors.**
- **The cost of the program.**

Using python

- IDLE has two modes: interactive and script
- Interactive mode immediately returns the results of commands you enter into the shell.
- In script mode, you will write a script and then run it.

- Under File, select New Window or press Ctrl + N. That will bring up a new window titled, “Untitled” to run the script mode.

- **We need to save our file before we can run it. With the extension .py**
- **Under Run, select Run Module**
- **The Shell is restarted every time you run a module in IDLE.**

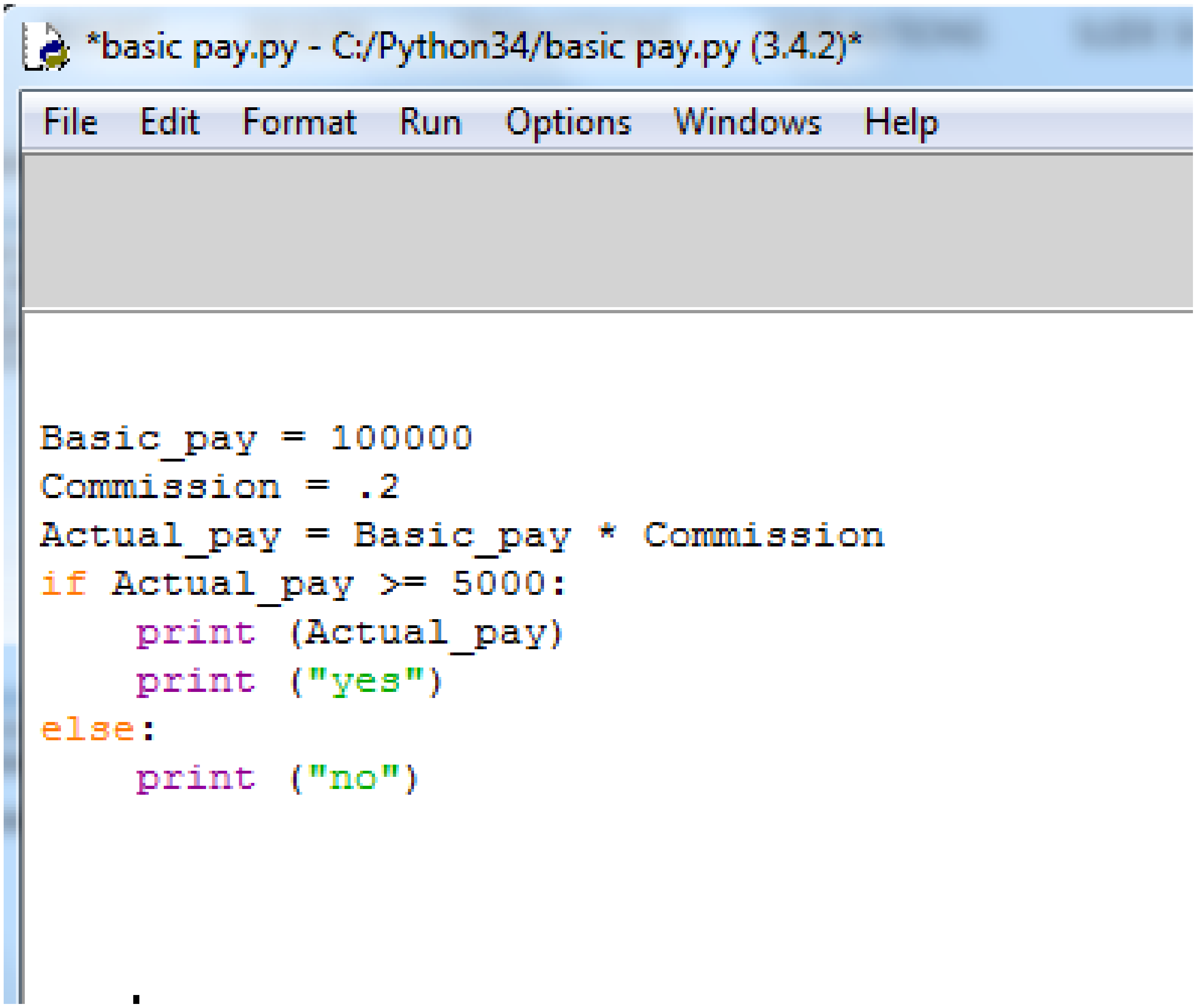
Declaring constants and variables

- Declaring a constant or Variable means that when you are writing a code you need to describe the variables or constants you are going to use before actually using them in the program.

There are two parts to a declaration:

- Declaration of a suitable name for the constant/variable.
- Declaration of the data type to be used

1. **Basic_pay as integer**
2. **Commission as a float**
3. **Actual_pay as a float**


basic pay.py - C:/Python34/basic pay.py (3.4.2)
File Edit Format Run Options Windows Help

```python
Basic_pay = 100000
Commission = .2
Actual_pay = Basic_pay * Commission
if Actual_pay >= 5000:
    print (Actual_pay)
    print ("yes")
else:
    print ("no")
```

```
Value1 = input("Enter first Value:")
Value2 = input ("Enter Value2:")
Product = Value1*Value2
Print (Product)
```

```
a = 2
b = 3
C = a + b
C = C + 1
Print (C)
```

Updating Variables

- This involves changing the value of the existing variable name in a function
- For example:
- X = 3
- Y = X + 2
- Y = 2 * Y
- X = Y – X
- Print (X, Y)

Input and storage instructions/commands

- The program/computer requires information to be entered into the system either by retrieving it from a stored file or data keyed in from the keyboard/terminal as the program is running.

Data input and storage instructions accept data that is entered into the computer and store the value in the location with the given variable names.
Common Commands used to input data are READ or INPUT.

The input function allows the user to key in data while the program is running when he/she is prompted to do so.

```
person = input (“Enter your name: “)
print (“Hello” + person)
```

Read statement

- The read statements obtains values from internally stored data.
- e.g.

```
        FOR  i = 1 TO 10
        READ  (i )
```

Example

- Write a program to enter the base and height of a triangle then calculate and print the area of the triangle.

1. **length = input (”What is the length of the Triangle?:”)**
2. **height = input (”What is the Height of the Triangle?:”)**
3. **area = (1/2 * length)* height**
4. **print (”The area of the triangle is” + area + ”squire cms”)**

Task

What are the input and storage instructions here?

- Read the price of an item and calculate the discount of 10%.
- Enter the name and year of birth of a person and calculate the person's age.
- Input the length of a side of a square tile and find the area.

Let's analyse these instructions to determine the inputs and what we need to store:

- Read and store the price of an item.
- Enter the name and year of birth of a person.
- Input the length of the side of a square tile.

Task

What are the processing instructions here?

- **Read the price of an item and calculate the new price after a 10% discount.**
- **Enter a temperature in degrees Celsius and convert it to degrees Fahrenheit.**
- **Input the name and year of birth and compute and print the age of a person.**

Let’s analyse these instructions to determine what needs to be processed:

- Calculate the new price after a 10% discount.
- Convert it to degrees Fahrenheit.
- Compute the age of a person.

Output instructions

Output instructions allow information to be displayed (written)on the screen. Statements that include key words like 'print', 'output', 'display', 'return' and 'write' indicate what data should be output to the screen.

Eg print(1+2)

What are the output statements here?

- **Enter the name and year of birth of a person and compute and display the age of the person.**

Outputting a string constant

In case of a string constant, the program displays the exact characters within the quotation marks.

- print ("Opiyo is my name“)
- print ("Time“)
- print ("Sum = 3 + 4“)

- Print is the function
- The argument is presented within round brackets
- The body of the text is within inverted commas (quotes)
- Quotes indicate to the interpreter should interpret it as mere text to display.

Exercise

Write a program that returns a sentence:
Hello world, nice to see you

- **Strings can be concatenated (more than one string can be combined and displayed as one**
- **Eg.**

```
>>> “Hello, ” + “World”
Hello, world
```

1. Use Python to calculate the number of minutes in a week.
2 Write a short program to print three lines: your name, your birth date, and your favorite color. The output should look something like this:

```
My name is Warren Sande.
I was born January 1, 1970.
My favorite color is blue.
```

- **If you want the text to include quotes surrounded by other text, need to be defined by mixing the quotes eg.To display**

He said to her, “how are you” before going to work.

```
>>> print (“He said, ‘how are you’ before going to work !”)
```

- **To put part of the text to another line use back slash followed with n for new line**

ie \n

e.g.

print (“ the on going seminar is about: \n Ecology \ n Biology”)

Text can be combined with variables

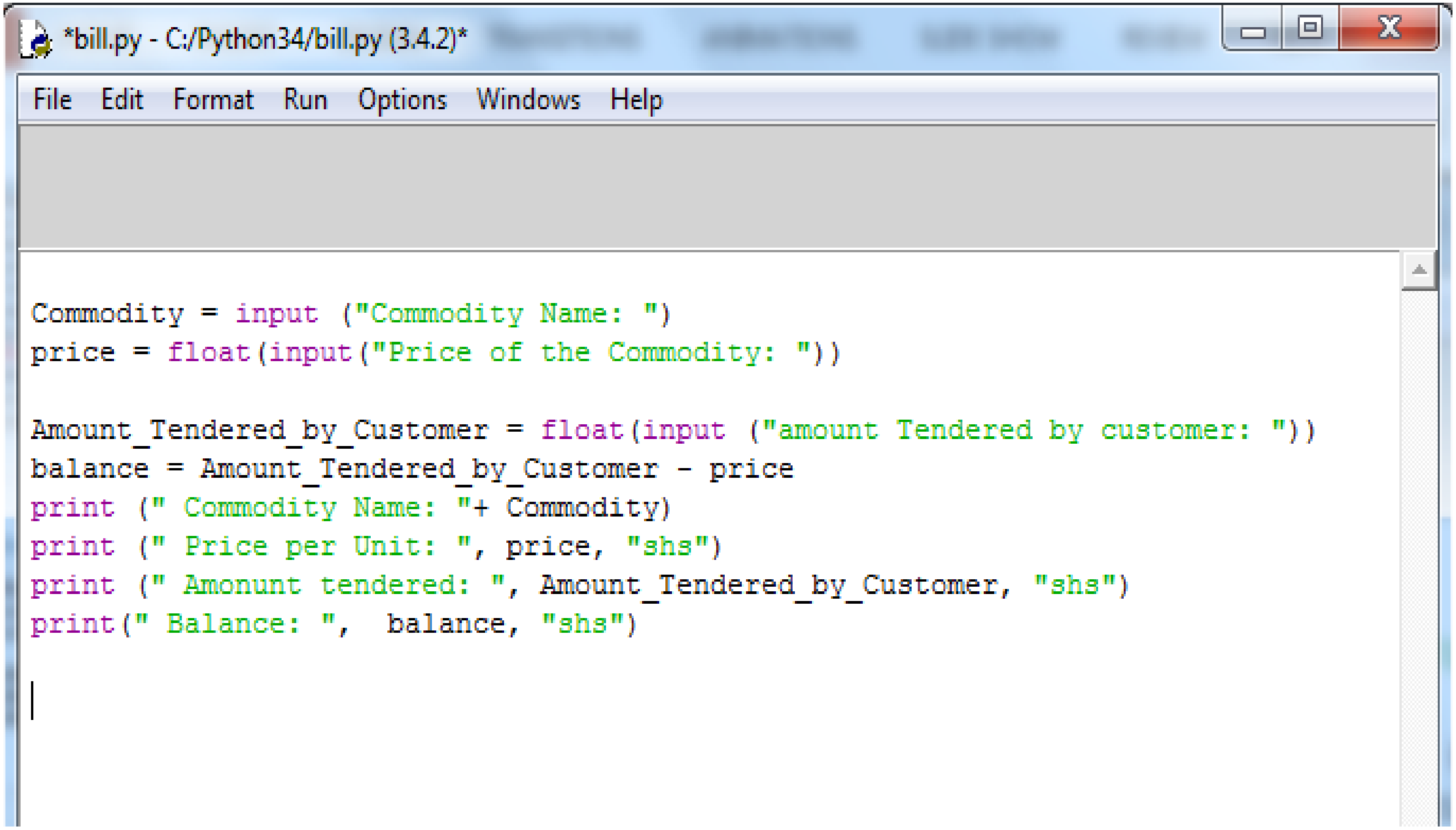
For example

```
a = 12
b = 23
sum = a + b
print (“ the sum of a and b is ” + sum )
```

Task

- Write an algorithm to print a conversion table of degrees Celsius to degrees Fahrenheit.

Write a program to Output a supermarket bill


bill.py - C:/Python34/bill.py (3.4.2)
File Edit Format Run Options Windows Help

```
Commodity = input ("Commodity Name: ")
price = float(input("Price of the Commodity: "))

Amount_Tendered_by_Customer = float(input ("amount Tendered by customer: "))
balance = Amount_Tendered_by_Customer - price
print (" Commodity Name: "+ Commodity)
print (" Price per Unit: ", price, "shs")
print (" Amonunt tendered: ", Amount_Tendered_by_Customer, "shs")
print(" Balance: ",  balance, "shs")
```

Let's analyse these instructions to determine what we need to output.

- Display the age of the person.
- Print a conversion table.

1 In interactive mode, make two variables, one for your first name and one for your last name. Then, using a single print statement, print your first and last names together.

2 Write a program that asks for your first name, then asks for your last name, and then
prints a message with your first and last names in it.

3 Write a program that asks for the dimensions (in feet) of a rectangular room, and then
calculates and displays the total amount of carpet needed to cover the room.

Structured program designs

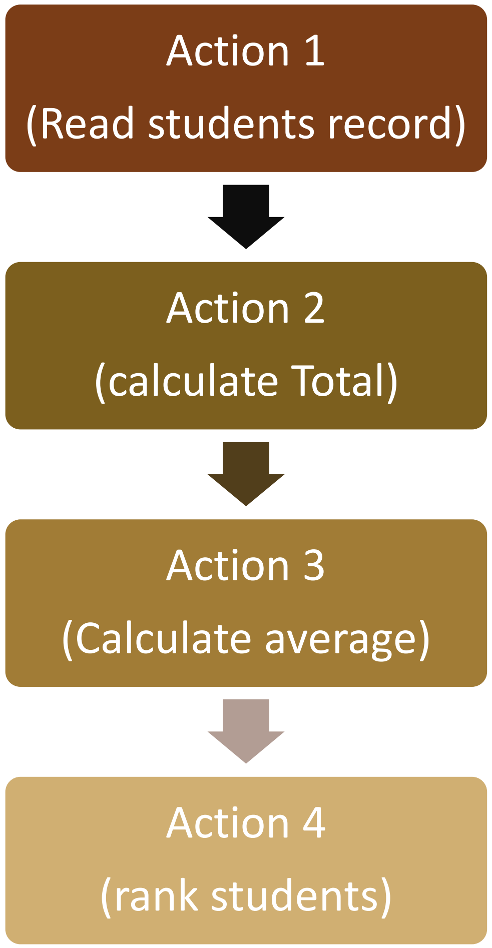
- This involves identifying the logical order of the procedure required to accomplish the task described in a given program module.

Four basic control structures are used to form the logic of a program in structured program design:
i)Sequence control,
ii)Selection control,
iii) Case control
iv) Iteration (loop).

The Sequence control structure

- The sequence structure is used to show a single action or one action followed sequentially by another e.g. reading a record, followed by calculating totals and averages and then printing the averages.

Sequence control structure



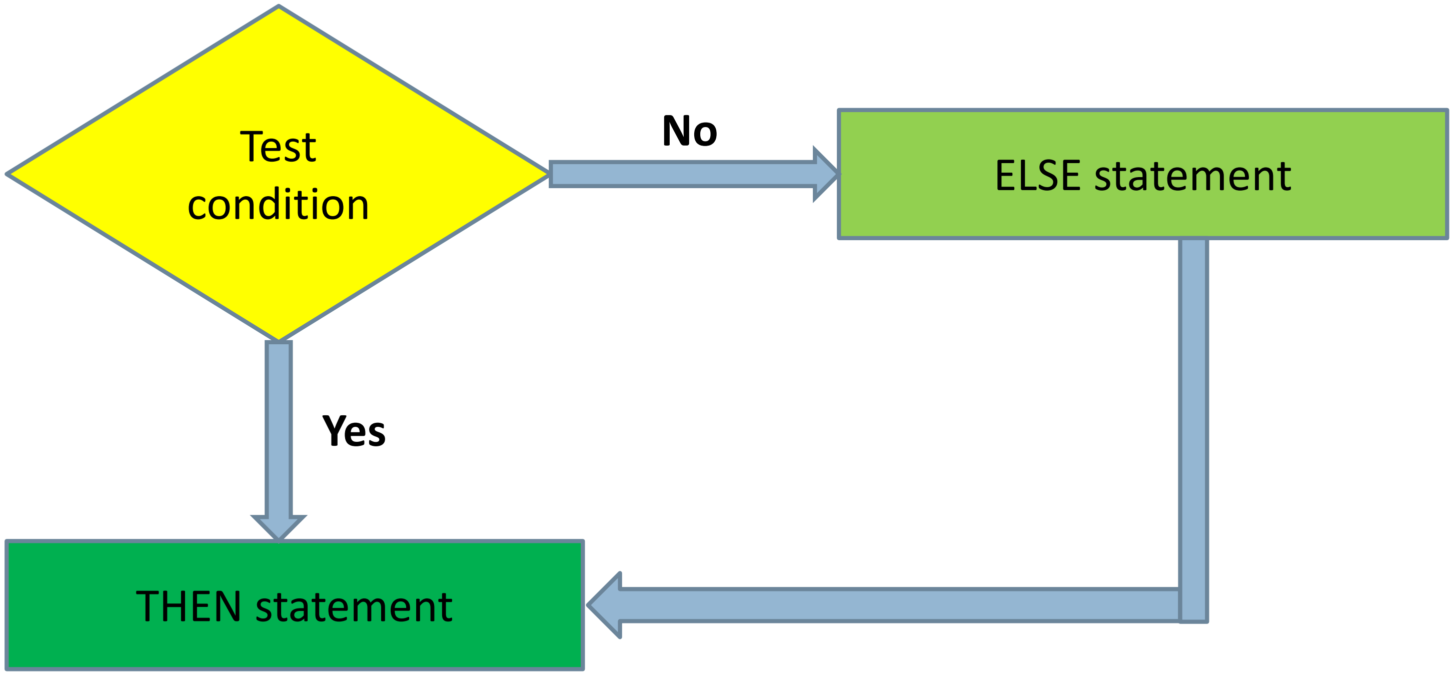
The Selection control structure (if-then-else)

- **Selection is a process that allows a computer program to compare values and then decide what course of action to take based on a certain condition.**

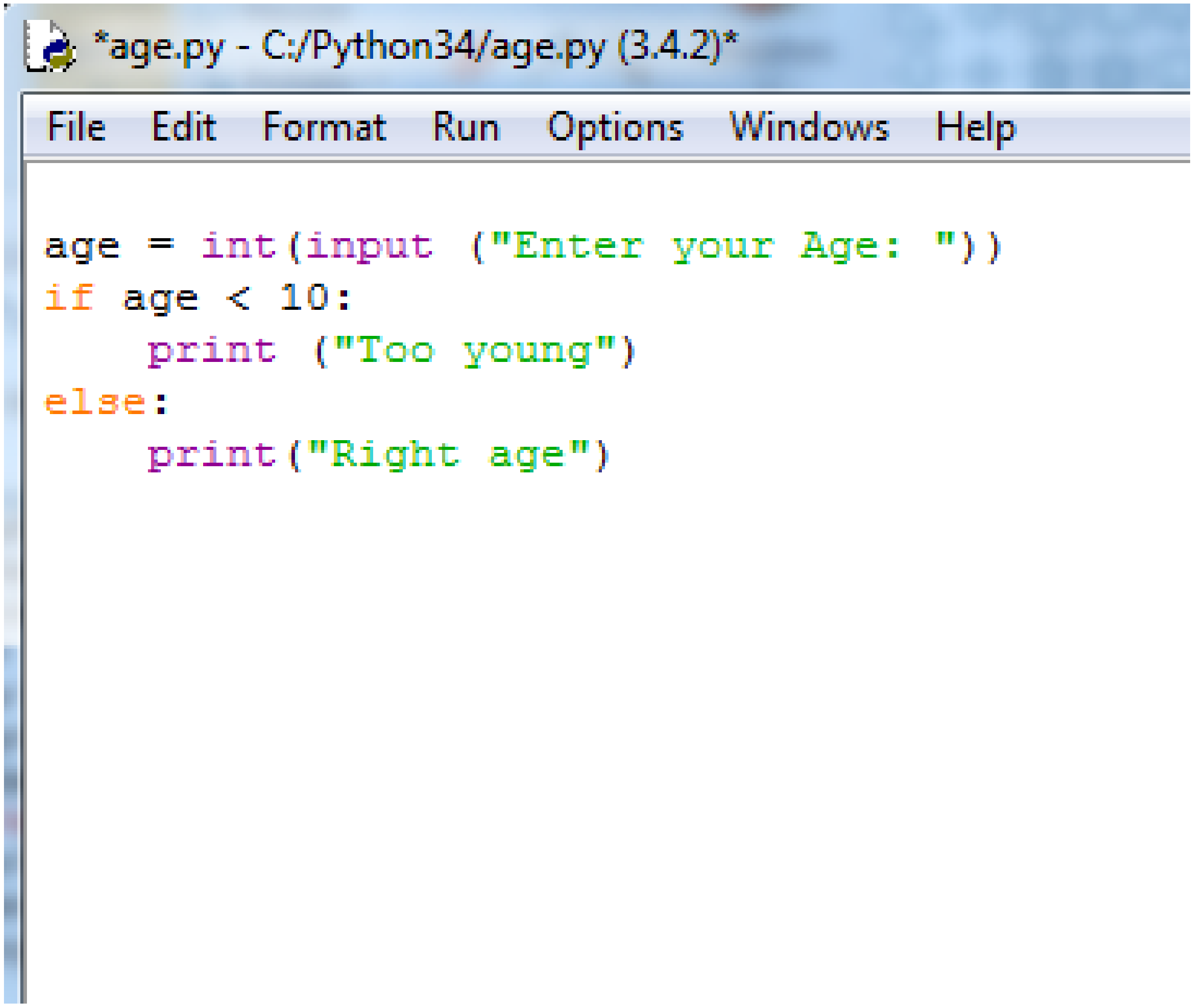
- The selection structure therefore is used to tell the program which action to take next, based on a certain condition; when the condition is evaluated, its result is either true or false : if the result of the condition is true, then one action is performed; if the result of the condition is false, a different or no action is performed.

- For example, a condition that if the average mark is 50 and above, the student is passed, or else, the student fails. i.e. if a condition is true, then do one thing, else do another.
- Because of the above, the selection structure is also known as the if-then-else control structure.

Selection control structure flowchart



```
IF score >= 60 THEN
    Comment = "Pass"
ELSE
    Comment = "Fail"
ENDIF
PRINT comment
```


age.py - C:/Python34/age.py (3.4.2)
File Edit Format Run Options Windows Help

```
age = int(input ("Enter your Age: "))
if age < 10:
    print ("Too young")
else:
    print("Right age")
```

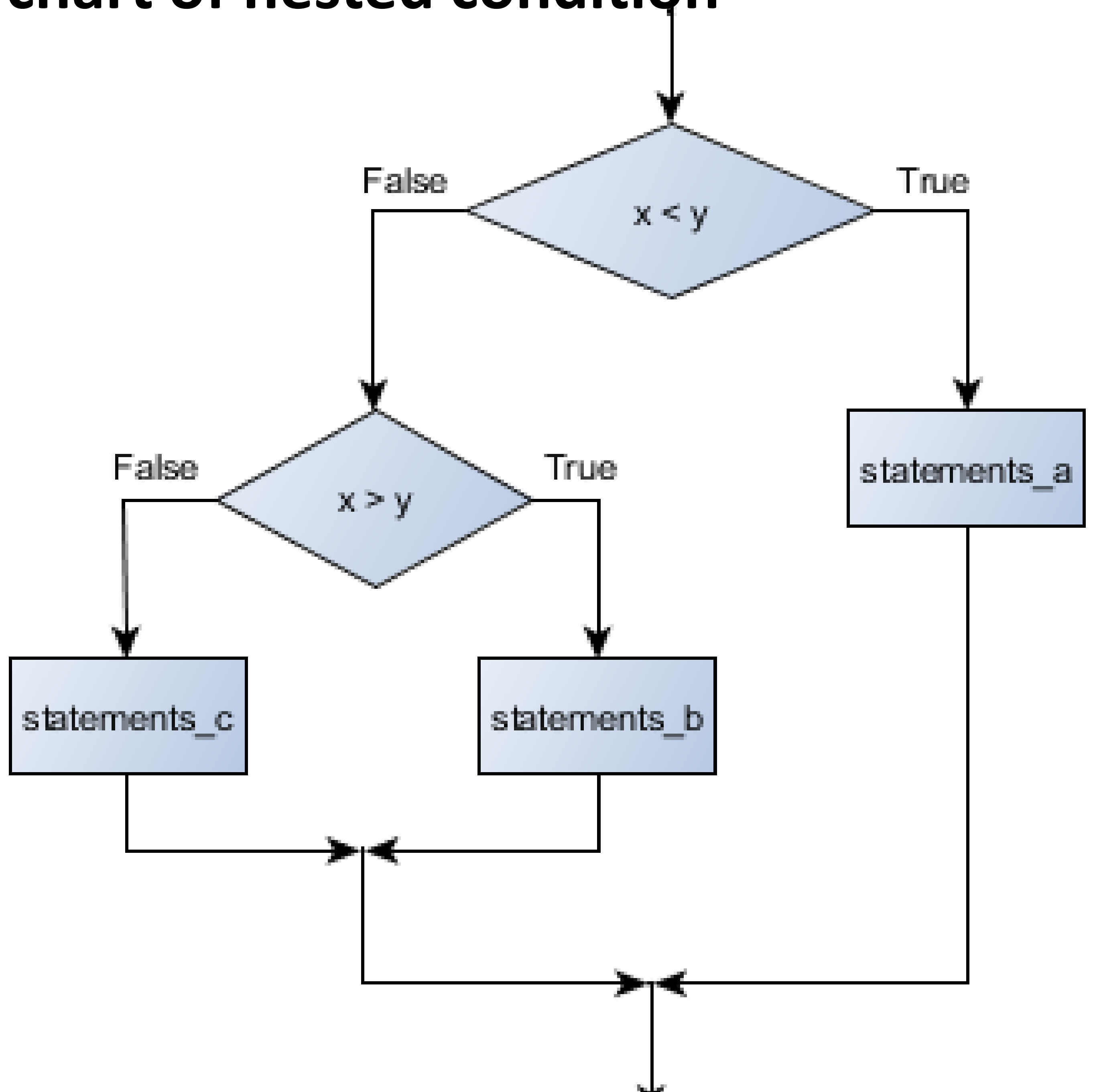
```
a = 8
b = 6
c = 9
If a > b:
c = a + c
Else:
c = c – a
Print (c)
```

Case control structure

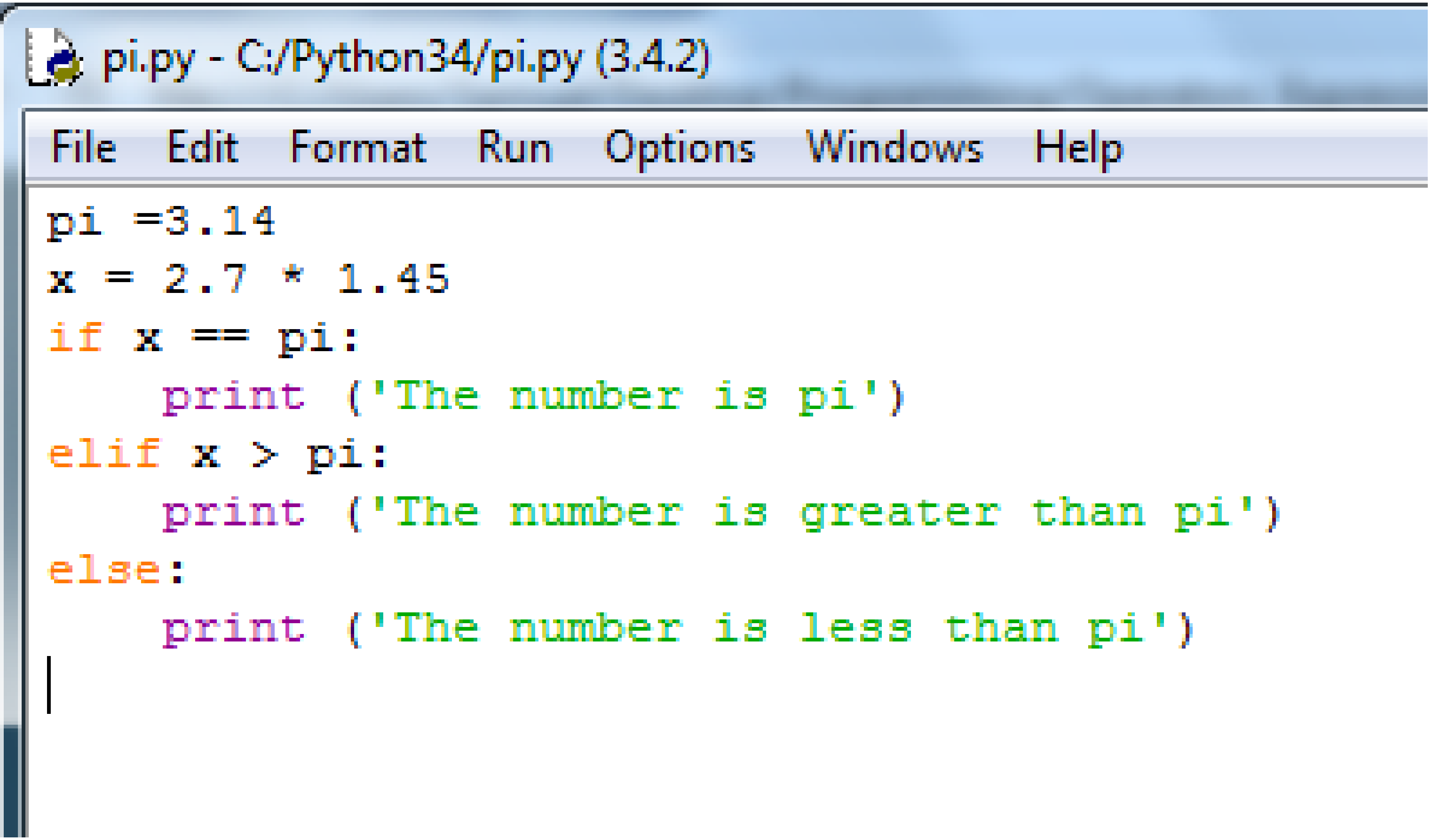
- This is a selection control structure used where a condition can yield more than two possibilities. Using
the If....Then....ElseIf statement.
- It allows several alternatives or cases to be presented, which saves the programmer the trouble of having to indicate a lot of separate IF...THEN...ELSE conditions.

- if the *condition* is True, the statements following *Then* are executed. If the *condition* is False, each *ElseIf* (if any) is evaluated in turn. If none of the ElseIf statements are True (or there are no ElseIf clauses), the statements following Else are executed.

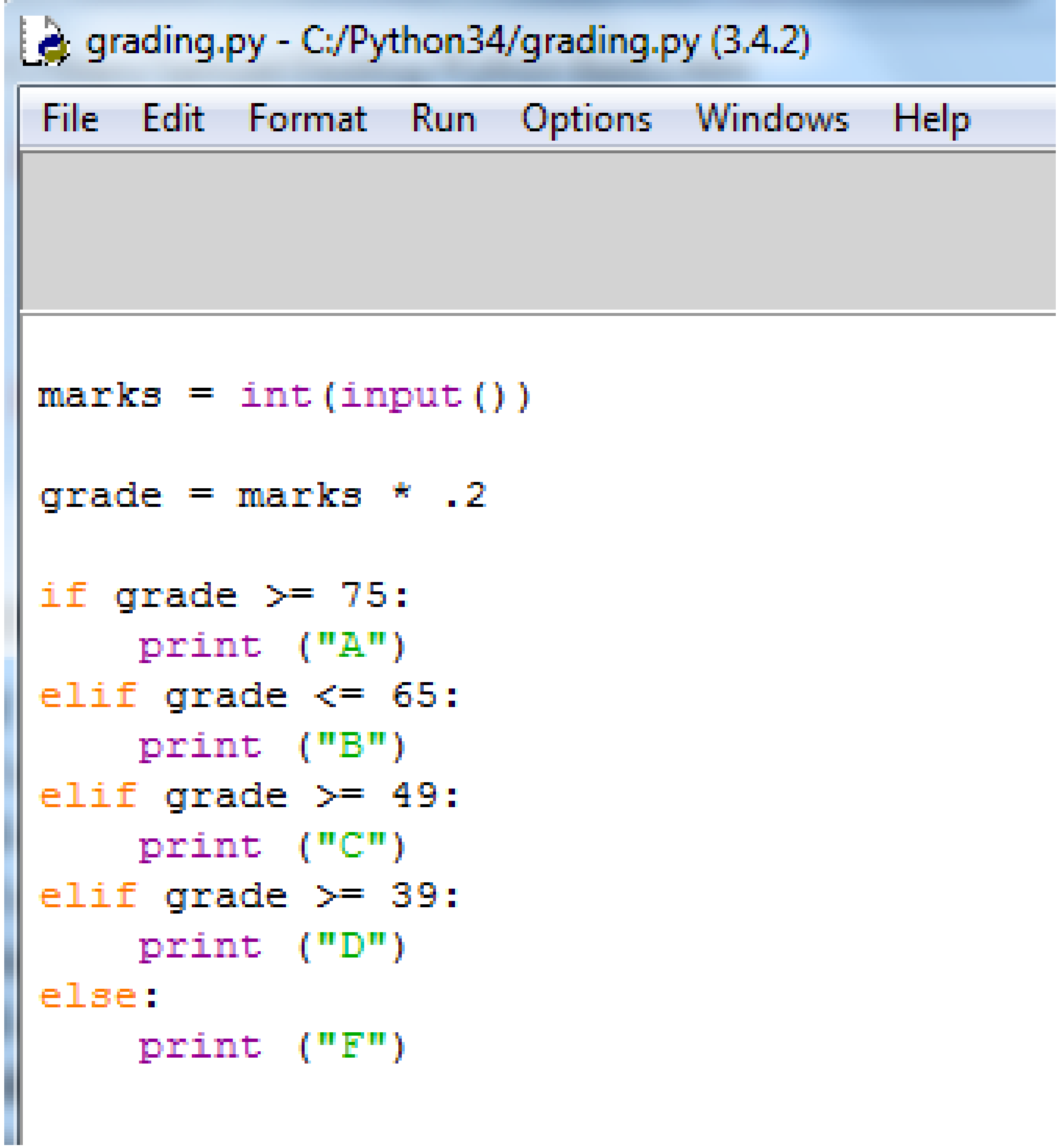
Flowchart of nested condition



```
if wage > 300000: then
 payment  = wage * .2
else:
if wage > 100000 : then
       payment = wage * .1
    else:
      payment = wage * 1
```


pi.py - C:/Python34/pi.py (3.4.2)
File Edit Format Run Options Windows Help

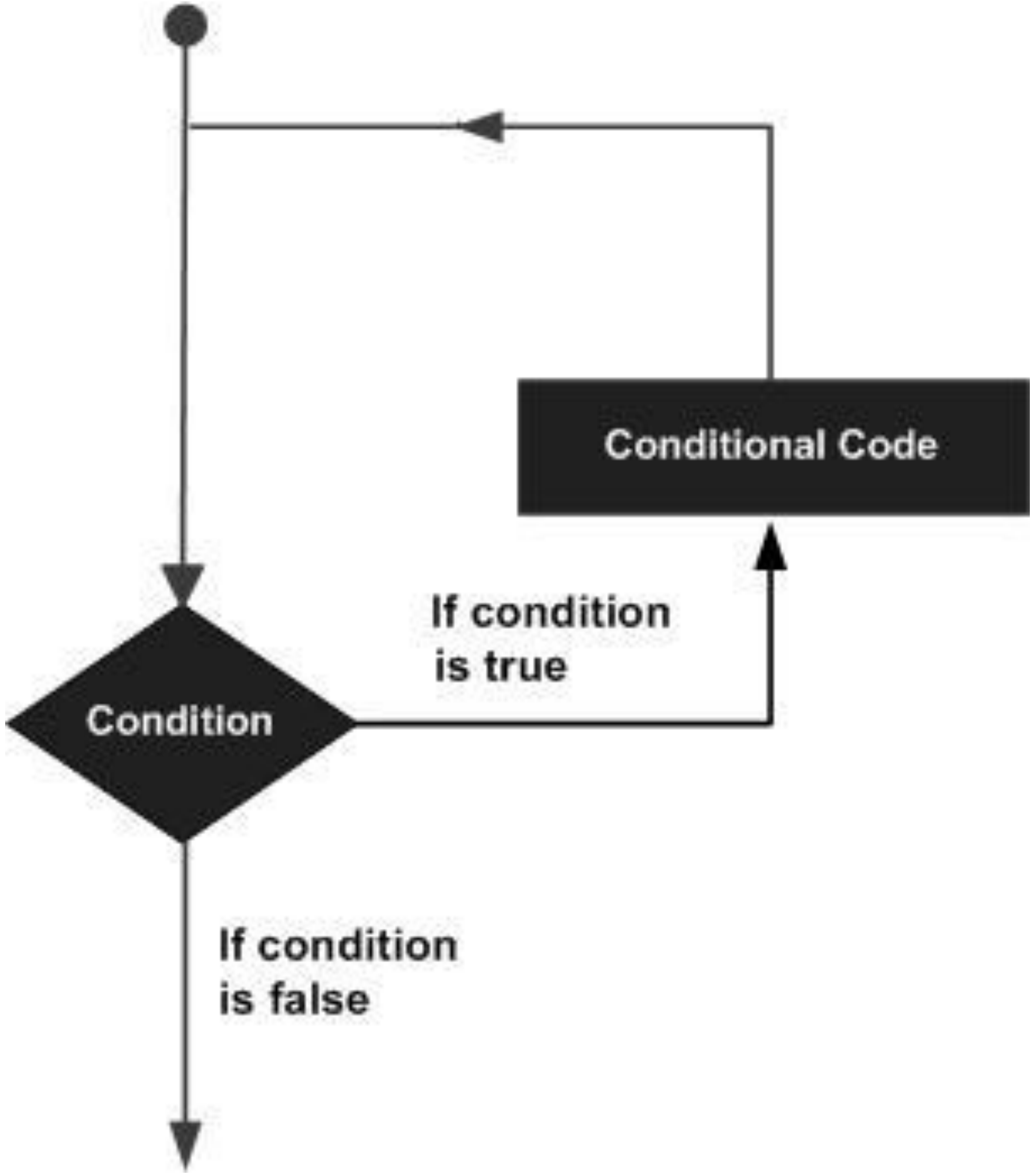
```
pi =3.14
x = 2.7 * 1.45
if x == pi:
    print ('The number is pi')
elif x > pi:
    print ('The number is greater than pi')
else:
    print ('The number is less than pi')
```


grading.py - C:/Python34/grading.py (3.4.2)
File Edit Format Run Options Windows Help

```
marks = int(input())

grade = marks * .2

if grade >= 75:
    print ("A")
elif grade <= 65:
    print ("B")
elif grade >= 49:
    print ("C")
elif grade >= 39:
    print ("D")
else:
    print ("F")
```

The Repetition/looping control structure (Iteration)

- It is a control structure in which an instruction is repeated as long as a certain condition remains true or until a given condition becomes true.


Conditional Code
If condition is true
Condition
If condition is false

- A loop may be a finite loop or infinite loop:
- A finite loop is one where the number of times a repetition is to be made is known

e.g.

```
□i = range (1,25)
□for numbers in i:
□    print (numbers)
```

- An infinite loop is indefinite because it is repeated without stopping.
- eg. If a condition never becomes false

- There are two forms of iteration;

i)the Conditional looping

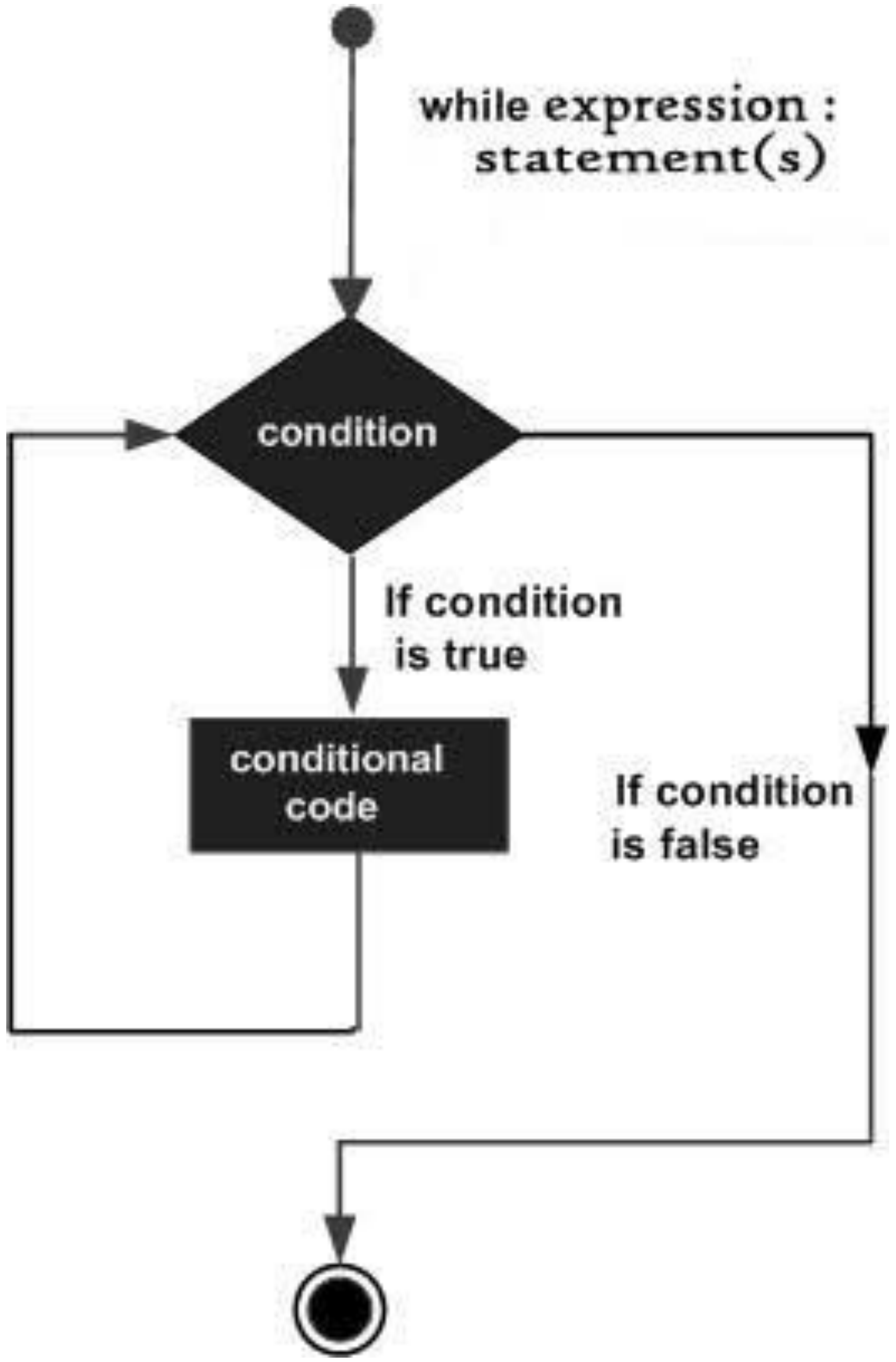
ii)the unconditional looping.

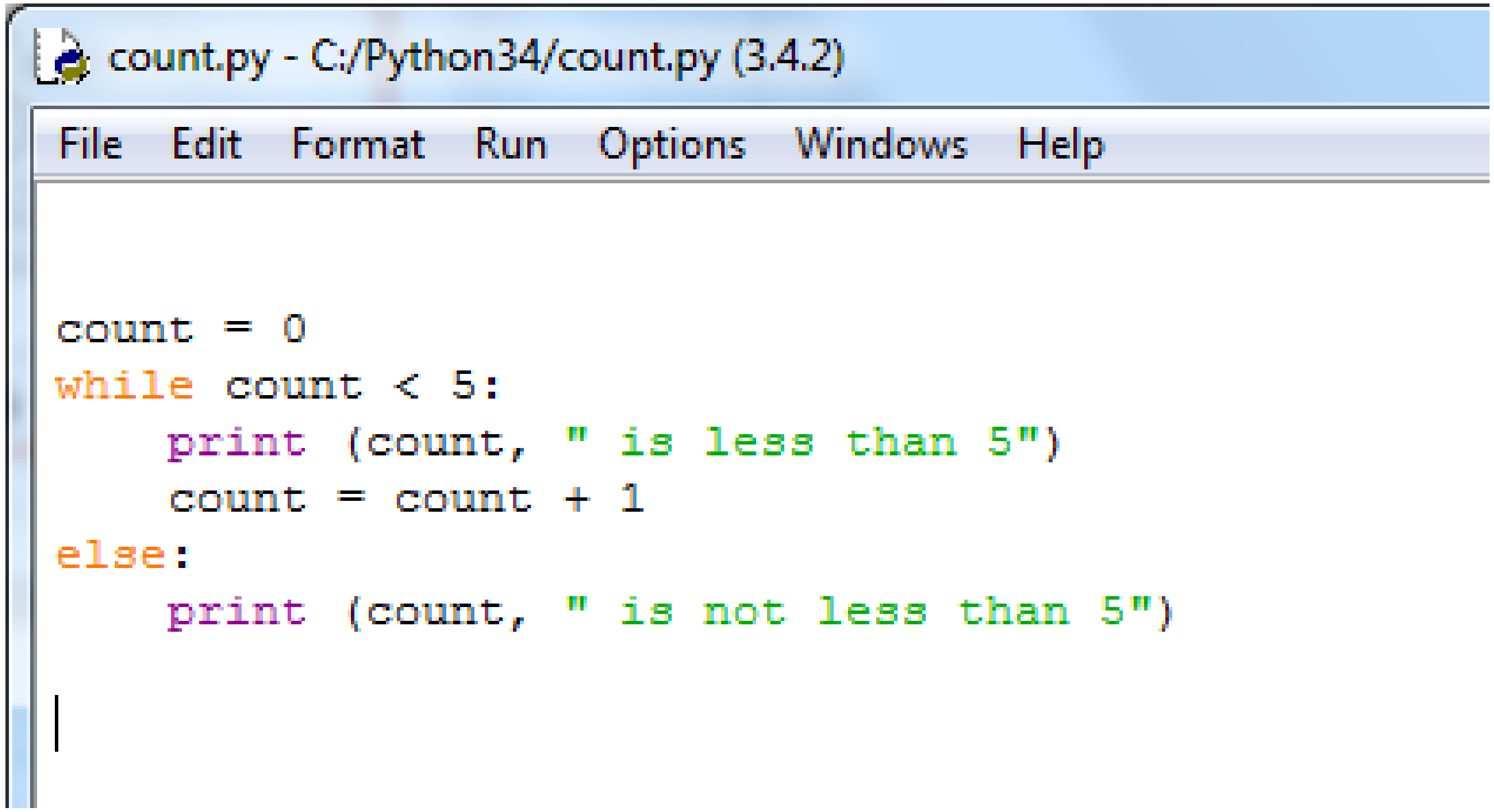
Conditional looping

- This looping is repeated until a specified condition is met or as long as a given condition remains true.
- It uses a selection process to decide whether or not to carry on a process.
- There are two forms of Conditional looping;
  - i)Do-while
  - ii) Do-until

The do-while repetition control structure

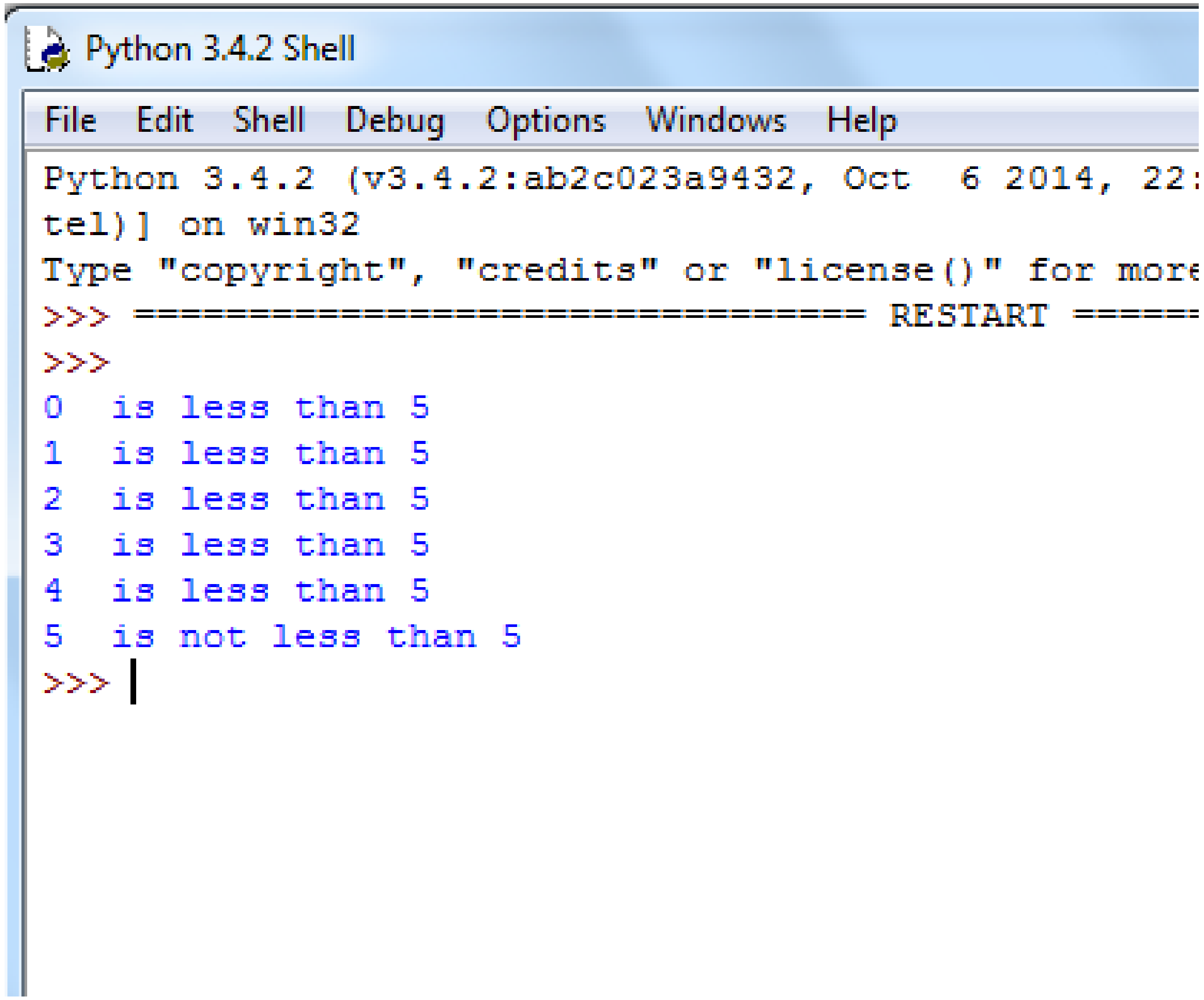
- **A while loop statement repeatedly executes the action(s) inside the loop as long as a given condition is true. i.e. until the condition becomes false.**


while expression :
statement(s)
condition
If condition is true
conditional code
If condition is false

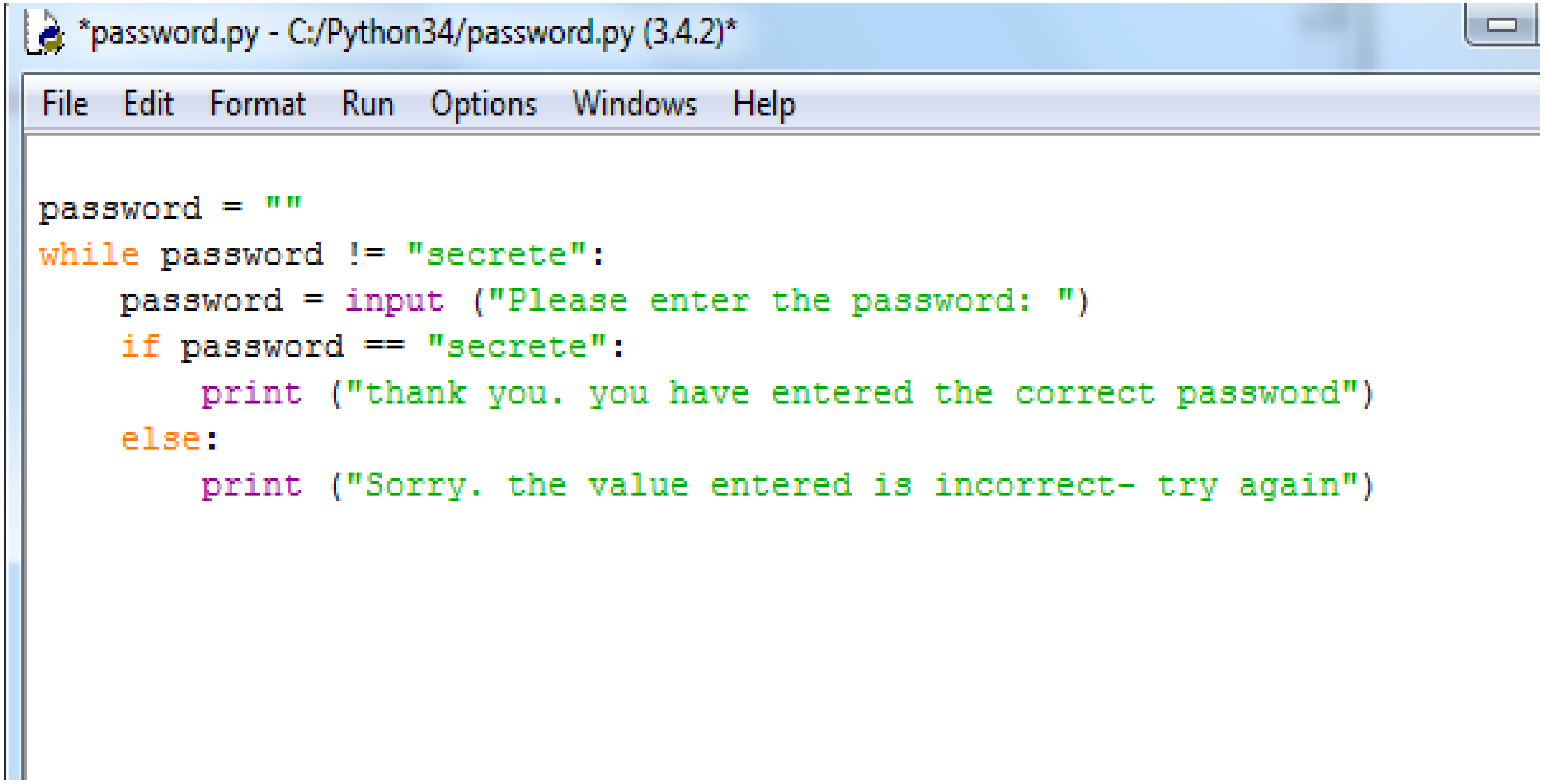

count.py - C:/Python34/count.py (3.4.2)
File Edit Format Run Options Windows Help

```
count = 0
while count < 5:
    print (count, " is less than 5")
    count = count + 1
else:
    print (count, " is not less than 5")
```

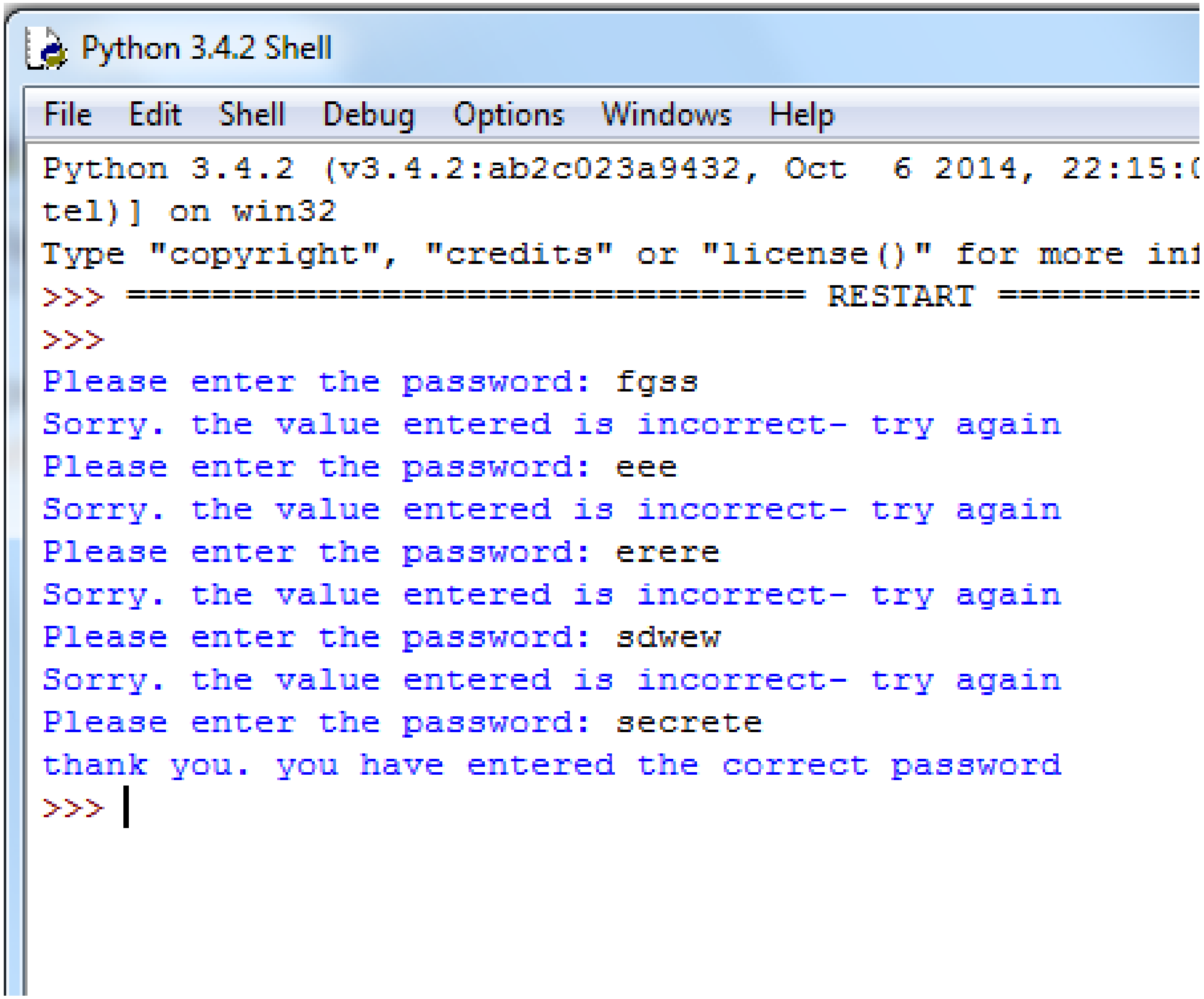
Having 0 as the first value (i.e. line 1: count = 0) but incremented by 1 (line 4: redefines the count variable as count = count + 1)



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:
tel)] on win32
Type "copyright", "credits" or "license()" for more
>>> ================================ RESTART ======
>>>
0  is less than 5
1  is less than 5
2  is less than 5
3  is less than 5
4  is less than 5
5  is not less than 5
>>> 
```


password.py - C:/Python34/password.py (3.4.2)
File Edit Format Run Options Windows Help

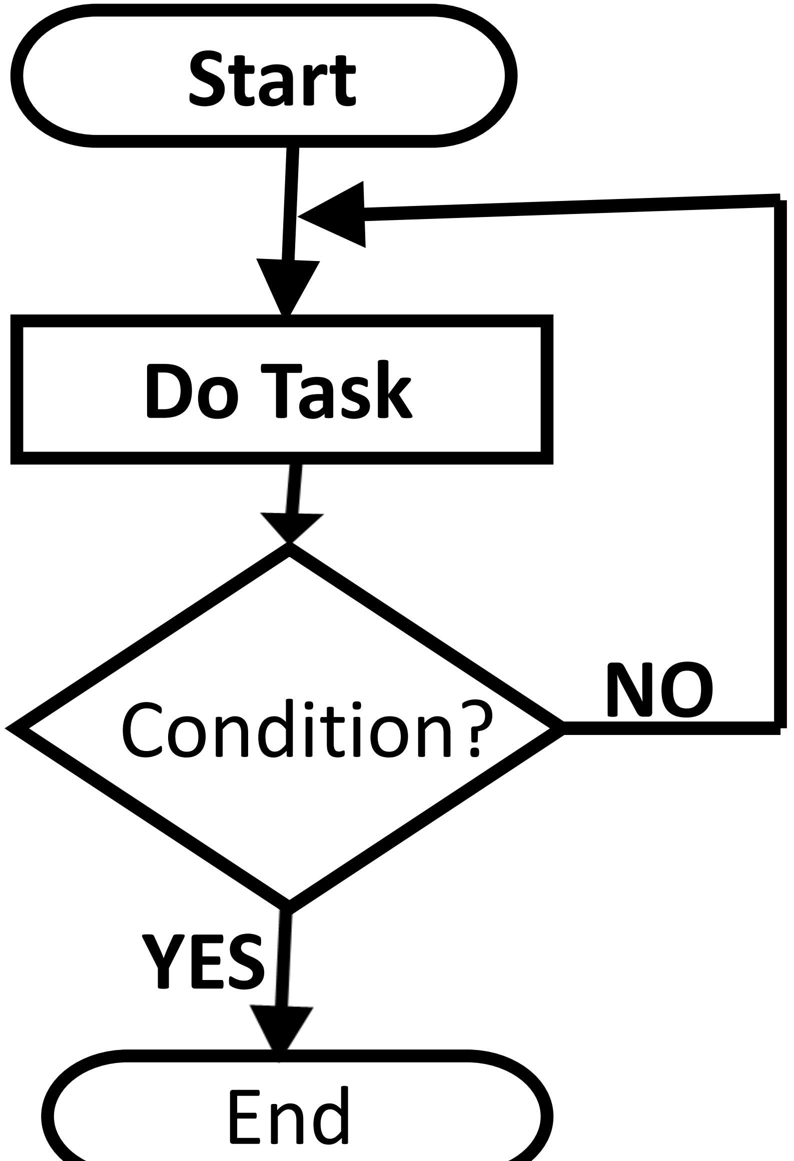
```
password = ""
while password != "secrete":
    password = input ("Please enter the password: ")
    if password == "secrete":
        print ("thank you. you have entered the correct password")
    else:
        print ("Sorry. the value entered is incorrect- try again")
```



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:0
tel)] on win32
Type "copyright", "credits" or "license()" for more inf
>>> ================================ RESTART ==========
>>>
Please enter the password: fgss
Sorry. the value entered is incorrect- try again
Please enter the password: eee
Sorry. the value entered is incorrect- try again
Please enter the password: erere
Sorry. the value entered is incorrect- try again
Please enter the password: sdwew
Sorry. the value entered is incorrect- try again
Please enter the password: secrete
thank you. you have entered the correct password
>>> 
```

The do-until selection control structure (Repeat Until loop)

- The loop continues until the specified condition is becomes true.
- For example, a robot may be programmed to move forward until it detects an object.

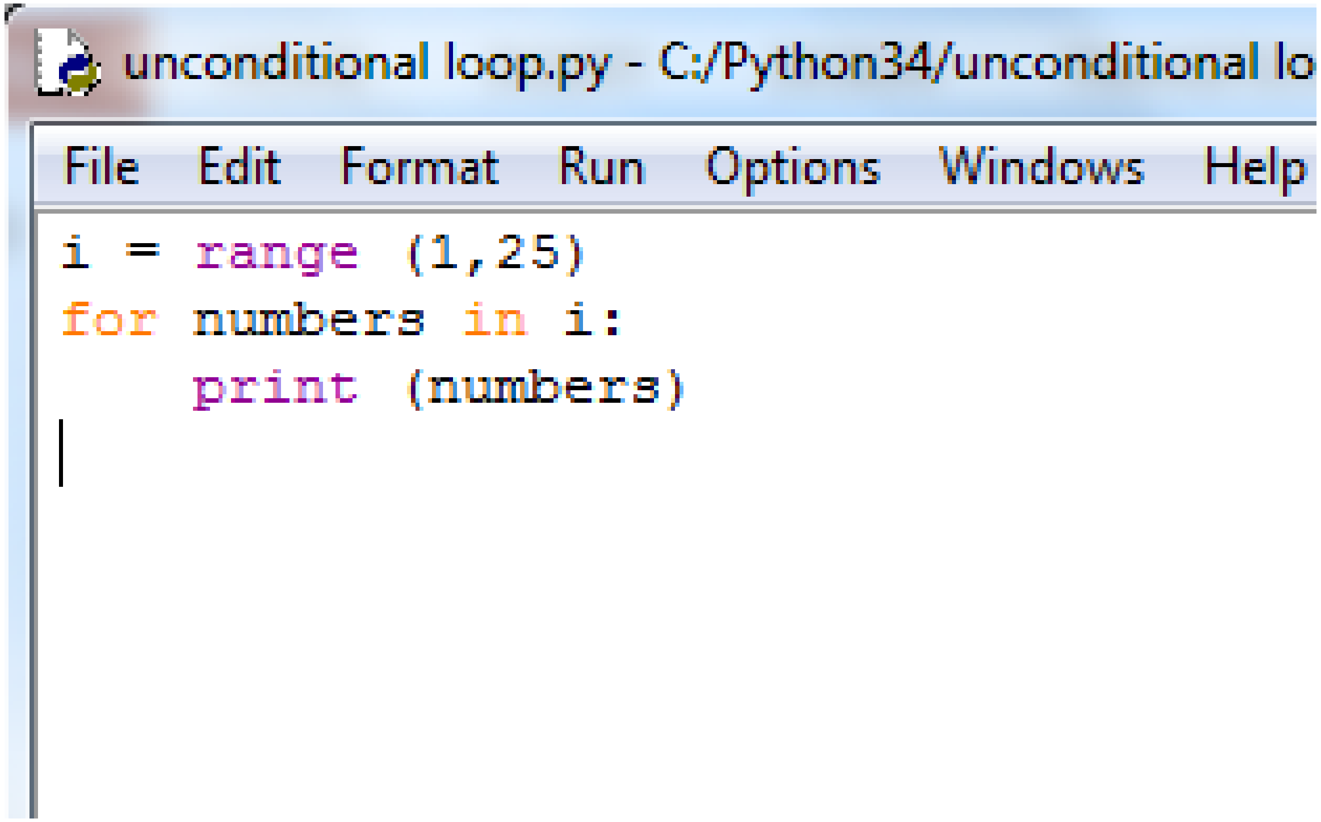

Start
Do Task
Condition?
NO
YES
End

Unconditional looping (For loop)

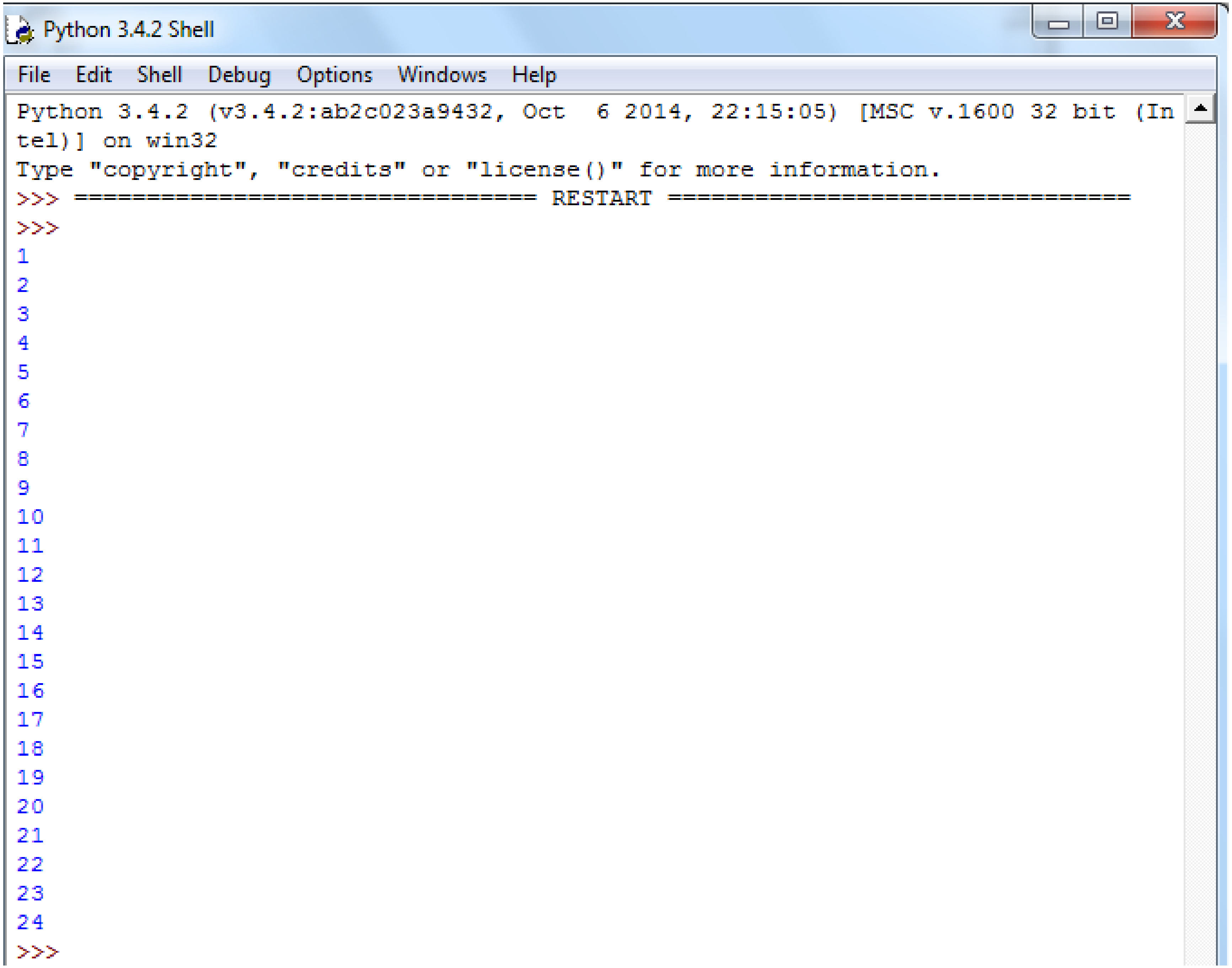
- This carries out a process a set number of times before it ends without conditions set. For example the following code will print out numbers in the range of 1 to 24 before it stops;
- i = range (1,25)
- for numbers in i:
-     print (numbers)

(Numbers is a variable name picked arbitrarily)

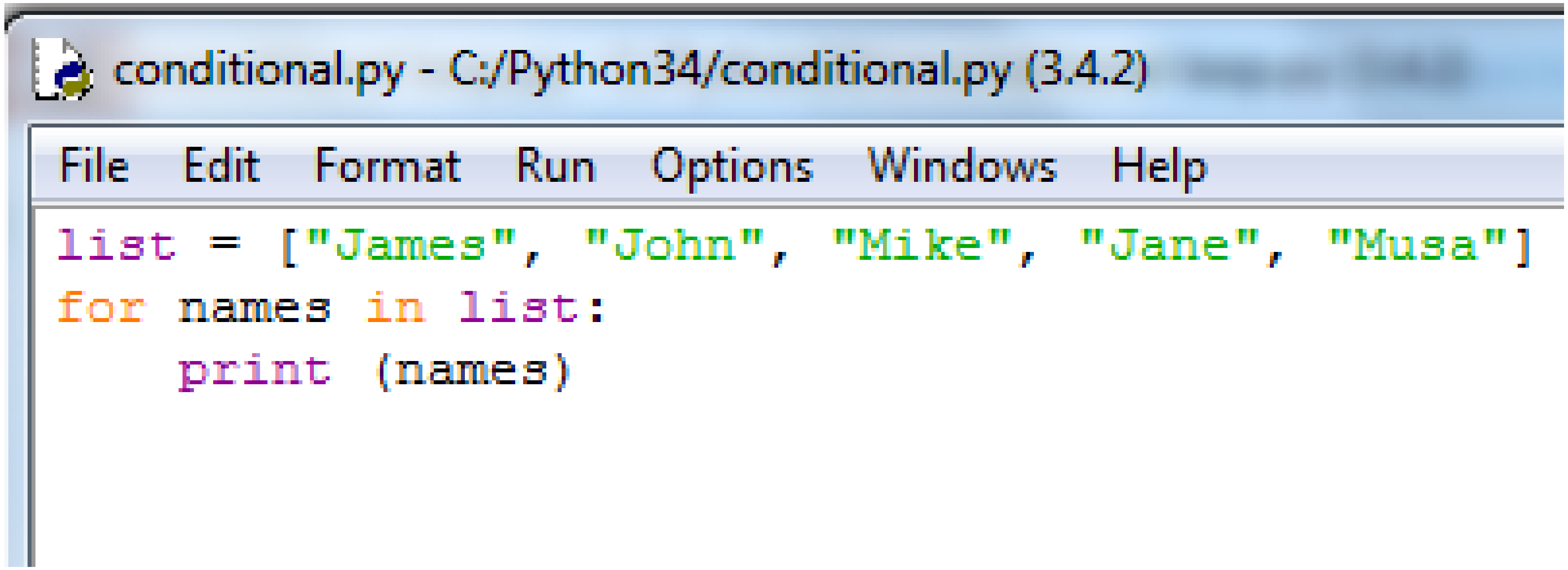
```
i = range (1,25)
for numbers in i:
    print (numbers)
```


unconditional loop.py - C:/Python34/unconditional lo
File Edit Format Run Options Windows Help

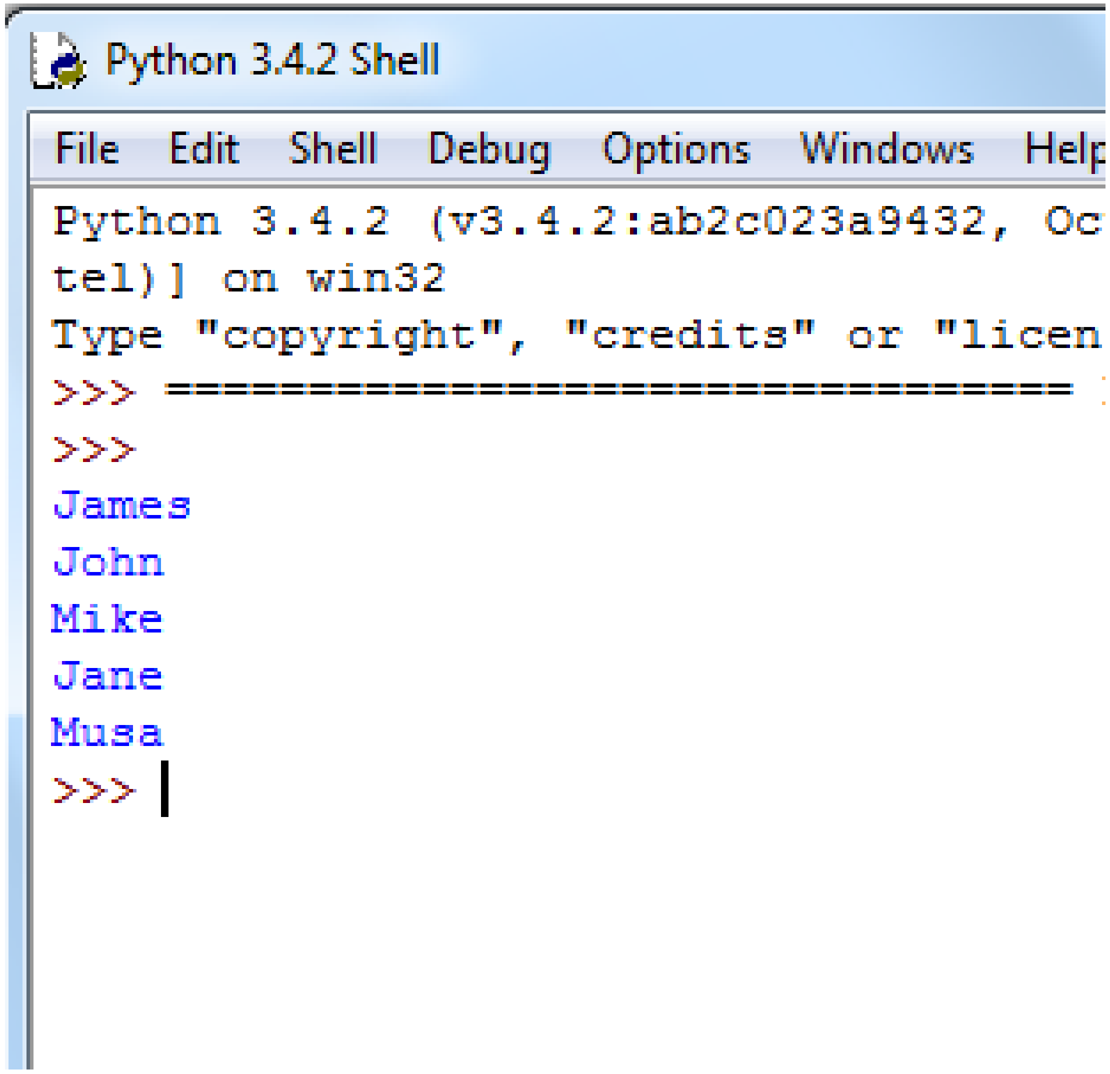
```
i = range (1,25)
for numbers in i:
    print (numbers)
```



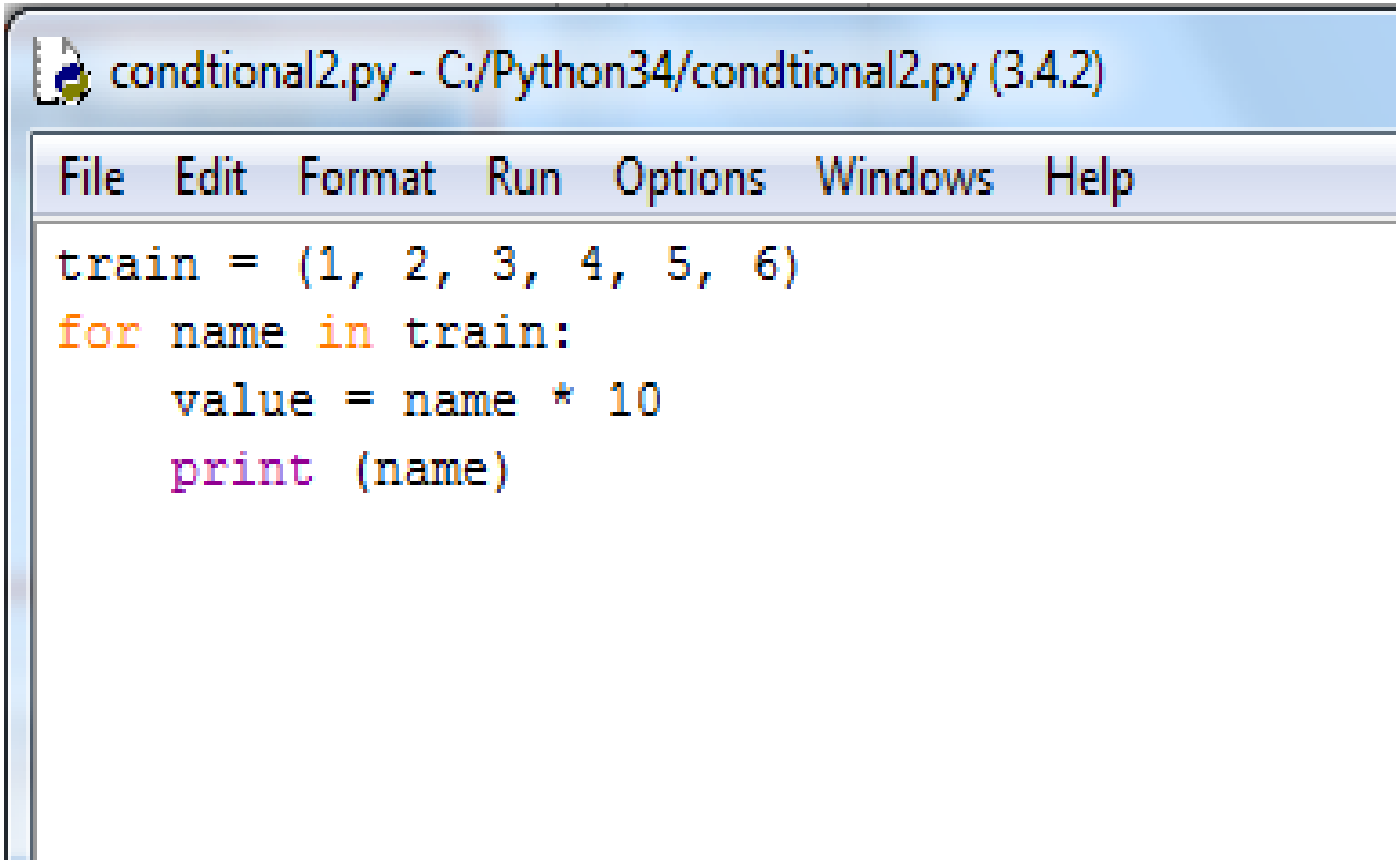
```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>>
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
>>>
```


conditional.py - C:/Python34/conditional.py (3.4.2)
File Edit Format Run Options Windows Help

```
list = ["James", "John", "Mike", "Jane", "Musa"]
for names in list:
    print (names)
```



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oc
tel)] on win32
Type "copyright", "credits" or "licen
>>> ================================
>>>
James
John
Mike
Jane
Musa
>>> 
```

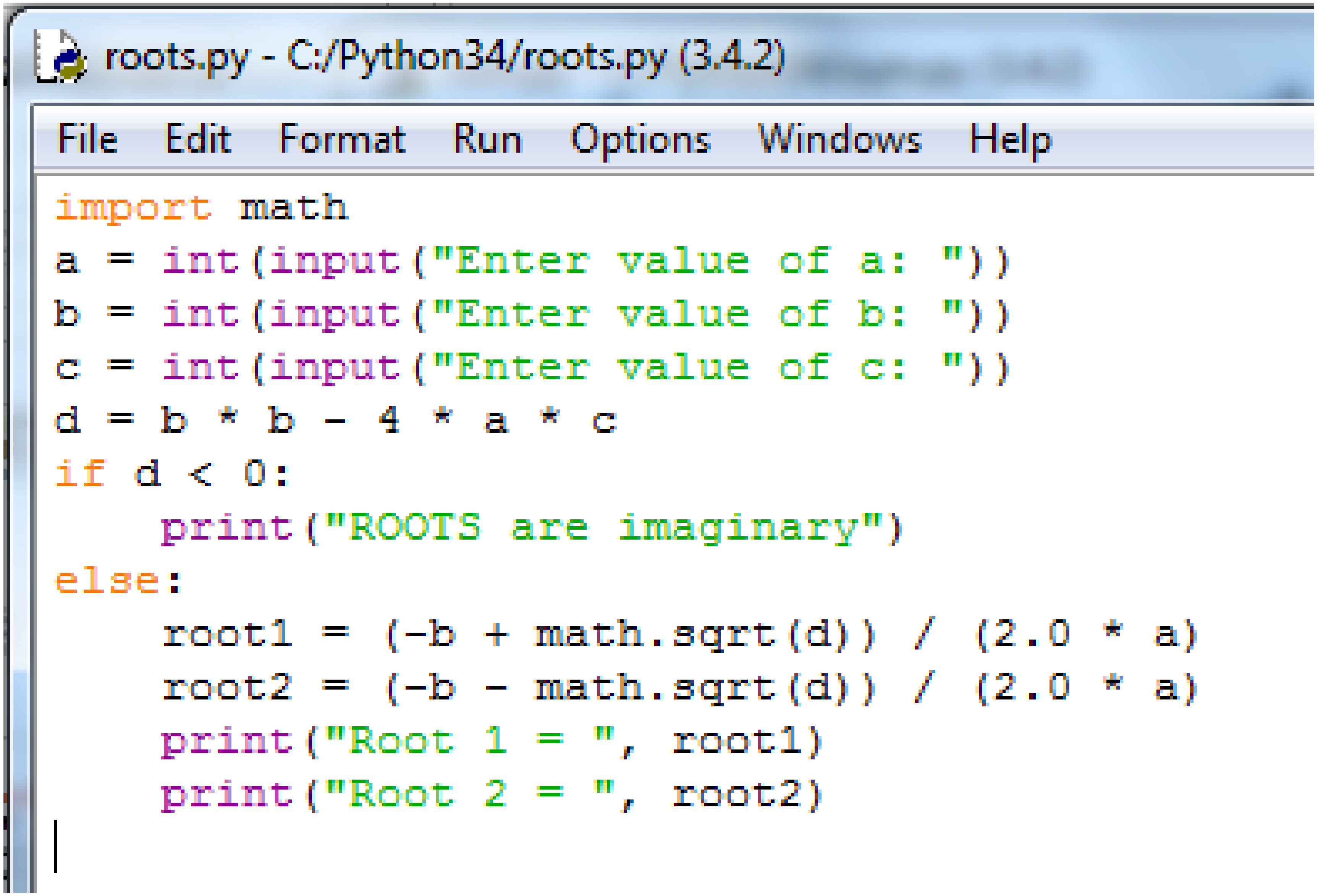

condtional2.py - C:/Python34/condtional2.py (3.4.2)
File Edit Format Run Options Windows Help

```
train = (1, 2, 3, 4, 5, 6)
for name in train:
    value = name * 10
    print (name)
```

Roots of a quadratic equation

a quadratic equastion is any equation in the form $ax^2 + bx + c = 0$ where $a \neq 0$
any quadratic equation can be can be solved with the formula
$x = (-b \pm \sqrt{(b^2 - 4ac)})\ /2a$
The discriminant, d of the equation is the expression b^2-4ac

```
import math
a = int(input("Enter value of a: "))
b = int(input("Enter value of b: "))
c = int(input("Enter value of c: "))
d = b * b - 4 * a * c
if d < 0:
   print("ROOTS are imaginary")
else:
   root1 = (-b + math.sqrt(d)) / (2.0 * a)
   root2 = (-b - math.sqrt(d)) / (2.0 * a)
   print("Root 1 = ", root1)
   print("Root 2 = ", root2)
```

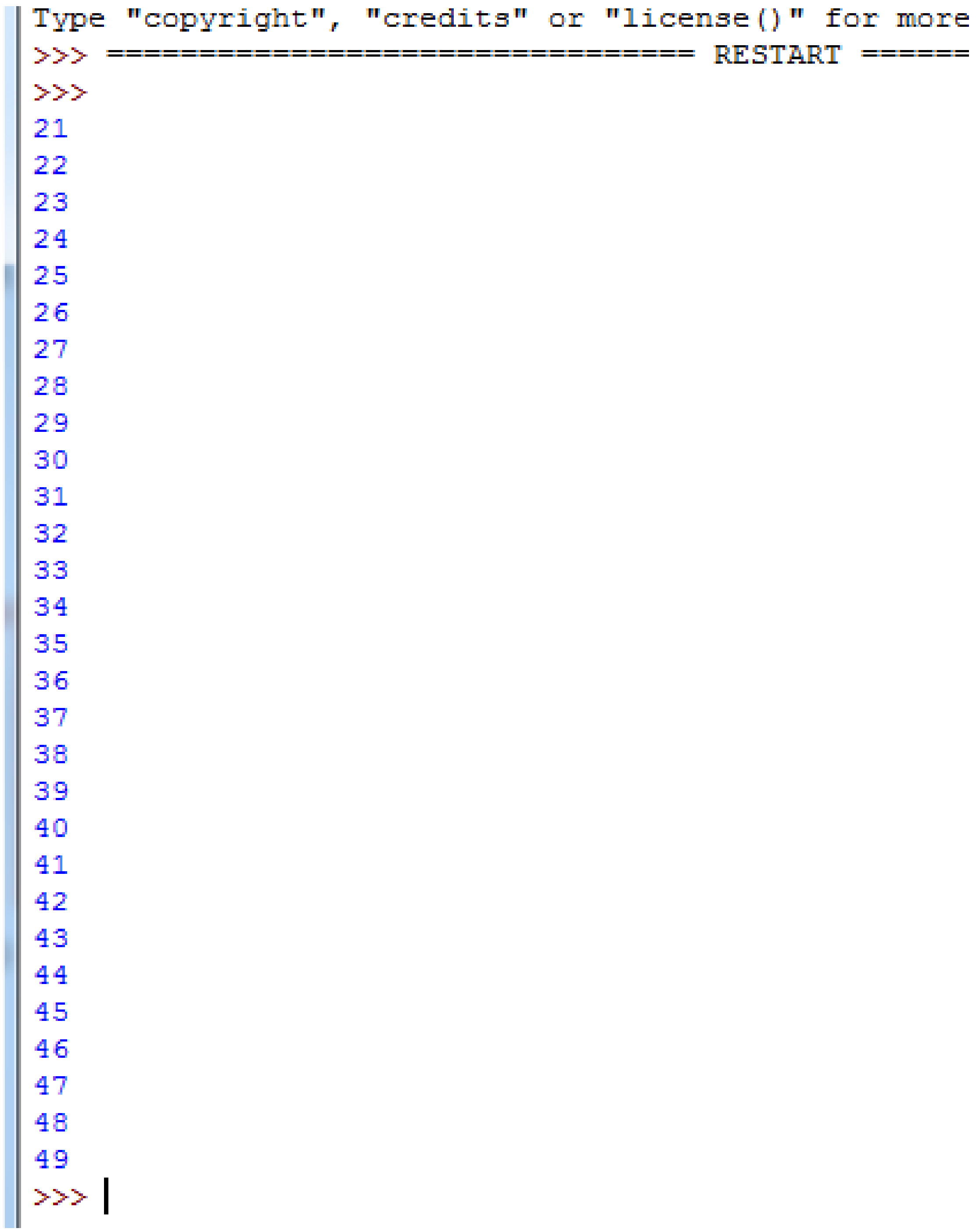

roots.py - C:/Python34/roots.py (3.4.2)
File Edit Format Run Options Windows Help

```
import math
a = int(input("Enter value of a: "))
b = int(input("Enter value of b: "))
c = int(input("Enter value of c: "))
d = b * b - 4 * a * c
if d < 0:
    print("ROOTS are imaginary")
else:
    root1 = (-b + math.sqrt(d)) / (2.0 * a)
    root2 = (-b - math.sqrt(d)) / (2.0 * a)
    print("Root 1 = ", root1)
    print("Root 2 = ", root2)
```

Branching

The use of a special statement that makes the computer jump to another statement other than the next sequential instruction. The common statement used include GOTO, IF/THEN, ELSE and FOR

```
i = range (2, 50)
for numbers in i:
 if (numbers > 20):
    print (numbers)
```



```
Type "copyright", "credits" or "license()" for more
>>> ================================ RESTART ======
>>>
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
>>>
```

Endless execution/infinite loop

A loop becomes infinite loop if a condition never becomes false.

```
var = 1
while var == 1 :
num = raw_input("Enter a number :")
print ("You entered: ", num")
print ("Good bye!")
```

Program testing and debugging

- This involves going through the program to identify and correct errors.
- It also involves creating test data to evaluate if the program generates the expected output.

Debugger

- This is a program used to test, detection and remove syntax and logic errors in a program.
- Debugging: a form of program testing OR the detection and removal of syntax and logic errors in a program.

- Debugging is a process that involves running the program using test data to discover and remove any errors in the program. There are three types of errors: syntax, logic and runtime errors.

- A syntax error is an error in the grammar of the programming language. These errors include typographical errors and incorrect use of the programming language.

- A logic error is an error in reasoning, such as the incorrect sequencing of instructions, and inconsistent comparisons and selection statements.

- Runtime errors occur during the execution/ running of the program.
- e.g. if the program is stuck in an infinite loop, such that the loop continues up to the time limit allowed by the computer set limit.

Documenting the Program

The programmer should provide a record about the program in the following aspects:

- What the program does.
- Instructions for program users on how to interact with the program.

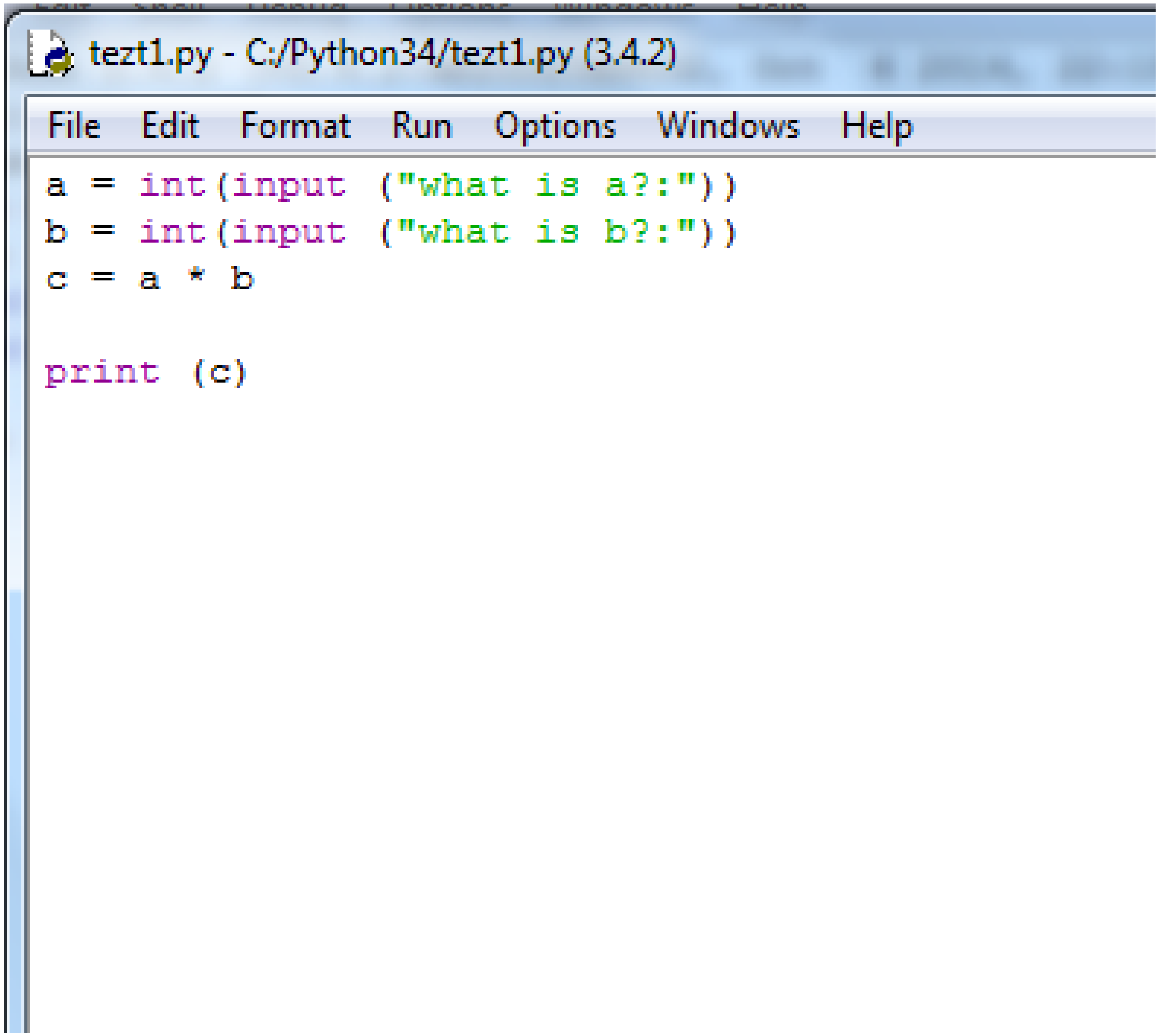
1 Solve the following problems either using interactive mode or by writing a small program:

a) Three people ate dinner at a restaurant and want to split the bill. The total is $35.27, and they want to leave a 15 percent tip. How much should each person pay?

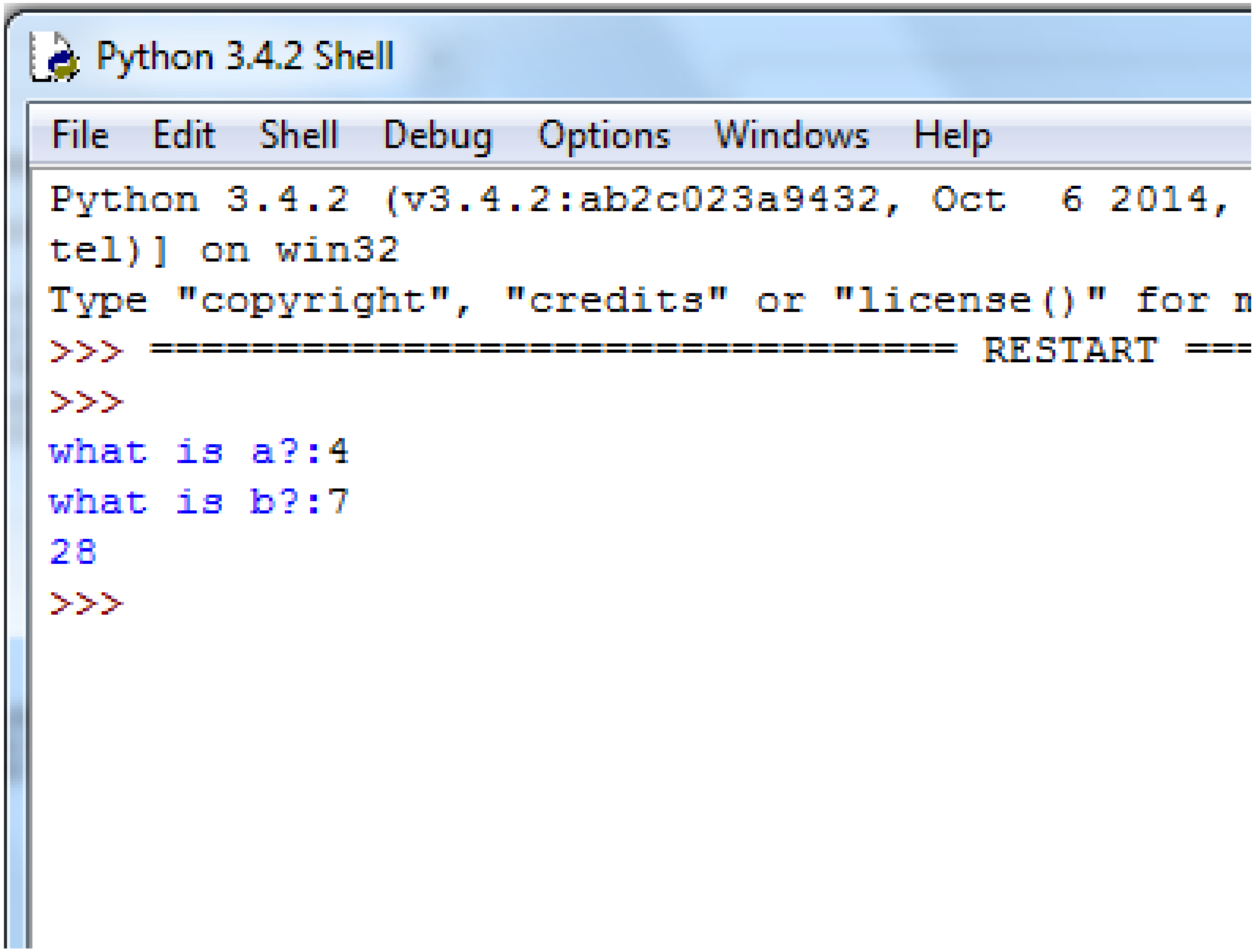
b) Calculate the area and perimeter of a rectangular room, 12.5 meters by 16.7 meters.

2 Write a program to convert temperatures from Fahrenheit to Celsius. The formula for that is: C = 5 / 9 * (F - 32).

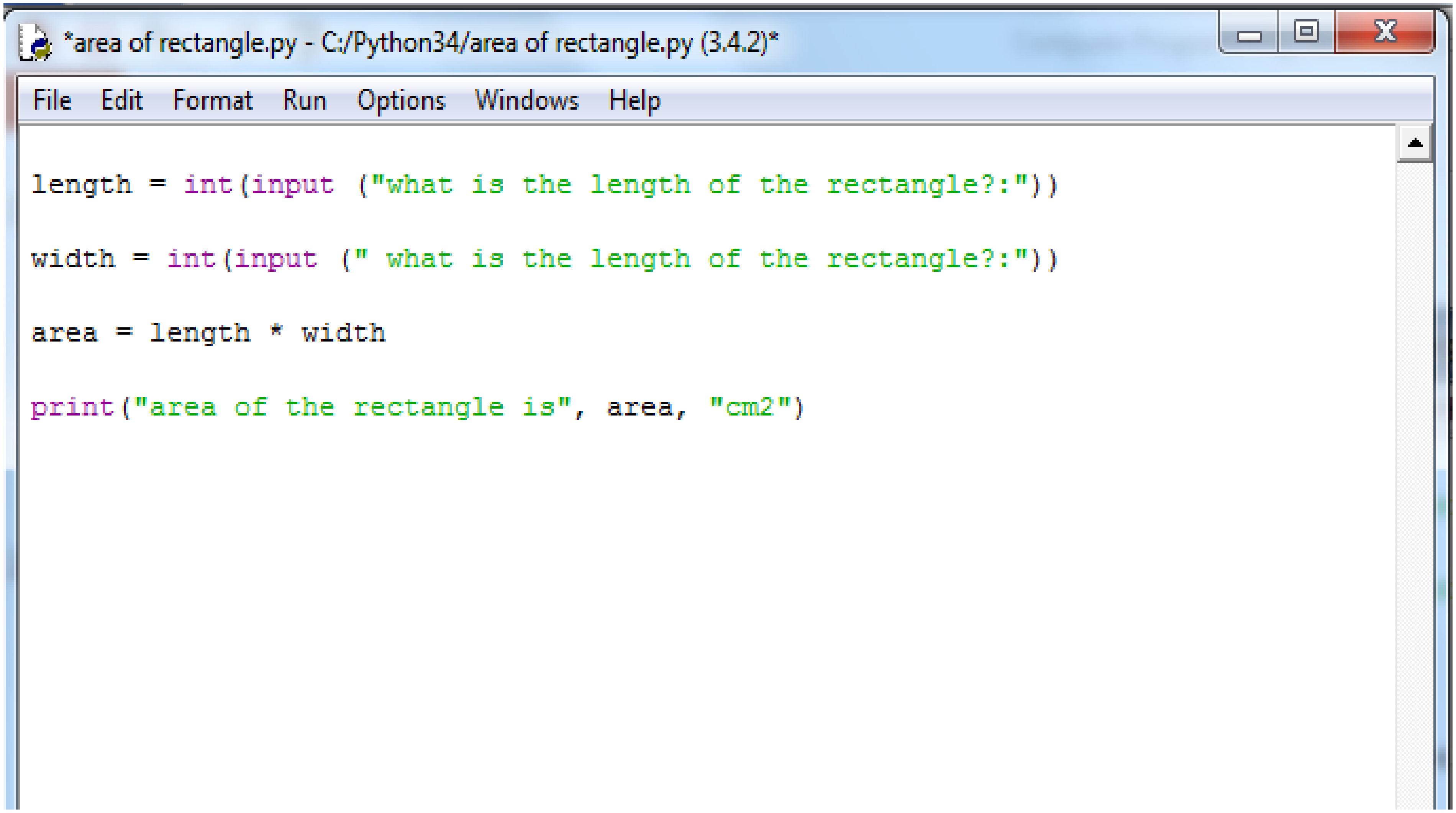
3 Do you know how to figure out how long it will take to get somewhere in a car? The formula
(in words) is “travel time equals distance divided by speed.” Make a program to calculate
the time it will take to drive 200 km at 80 km per hour and display the answer.


tezt1.py - C:/Python34/tezt1.py (3.4.2)
File Edit Format Run Options Windows Help

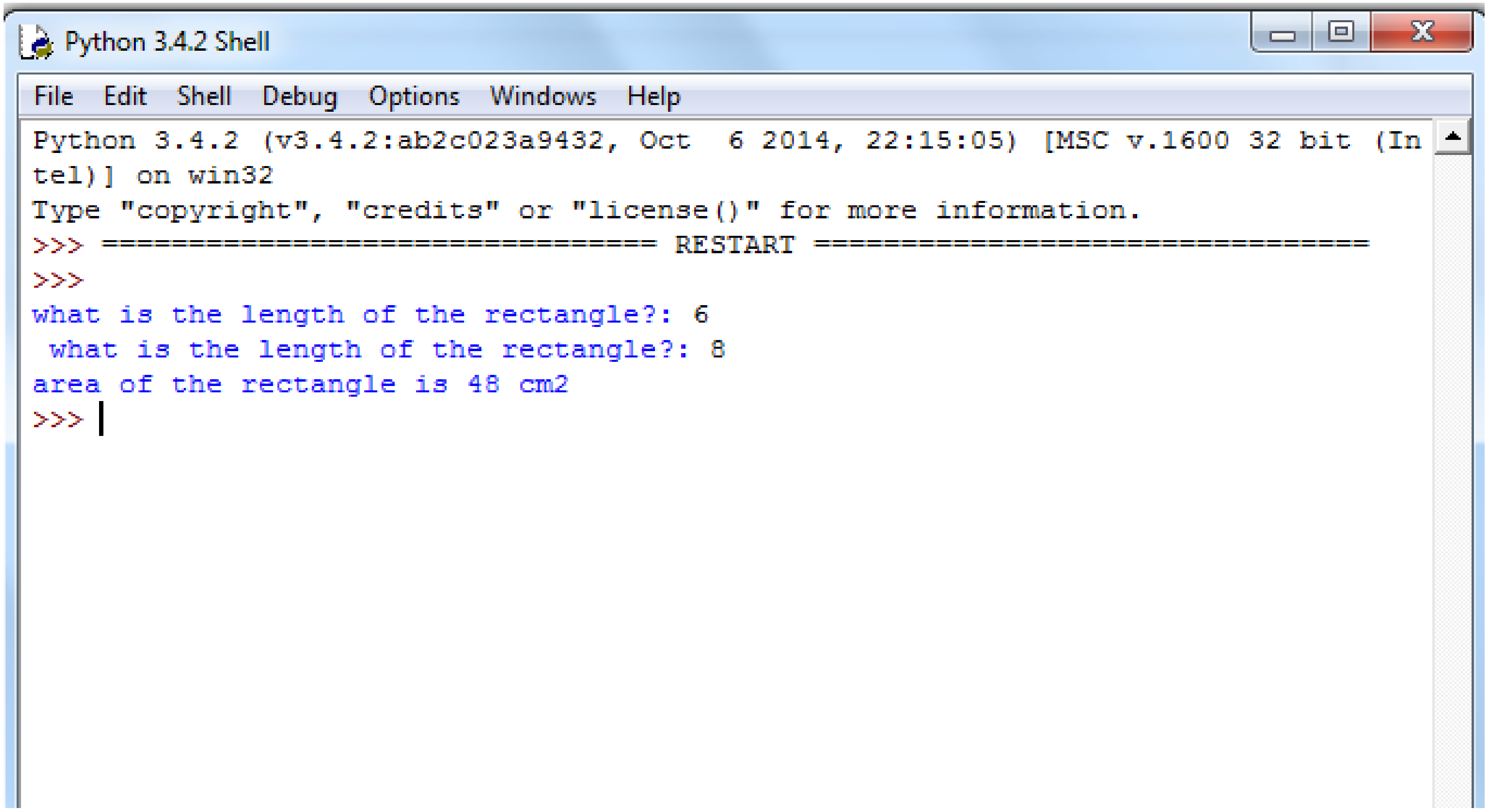
```
a = int(input ("what is a?:"))
b = int(input ("what is b?:"))
c = a * b

print (c)
```



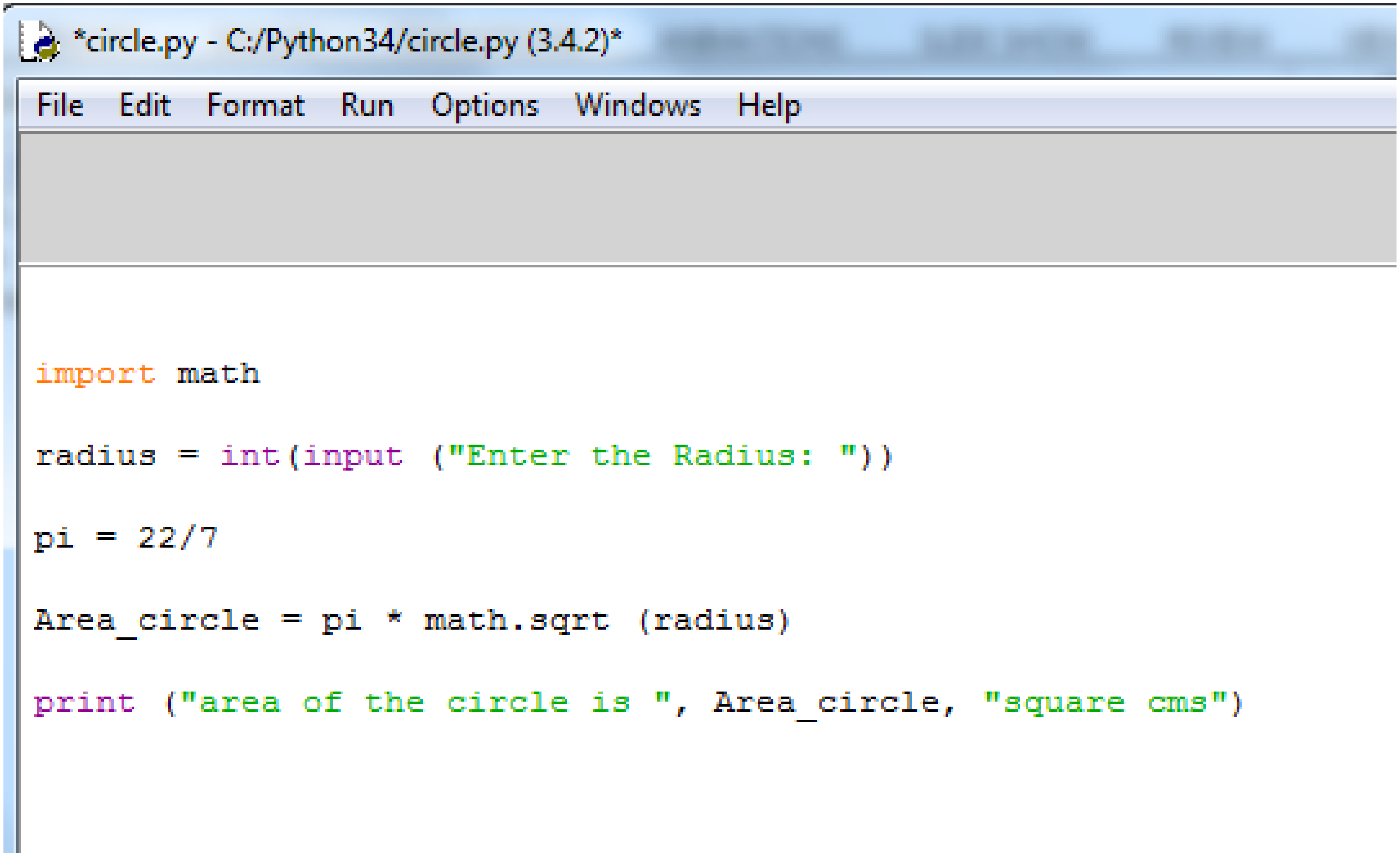
```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014,
tel)] on win32
Type "copyright", "credits" or "license()" for m
>>> ================================ RESTART ===
>>>
what is a?:4
what is b?:7
28
>>>
```


area of rectangle.py - C:/Python34/area of rectangle.py (3.4.2)
File Edit Format Run Options Windows Help

```
length = int(input ("what is the length of the rectangle?:"))

width = int(input (" what is the length of the rectangle?:"))

area = length * width

print("area of the rectangle is", area, "cm2")
```



```
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct  6 2014, 22:15:05) [MSC v.1600 32 bit (In
tel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ================================ RESTART ================================
>>> 
what is the length of the rectangle?: 6
 what is the length of the rectangle?: 8
area of the rectangle is 48 cm2
>>> 
```


circle.py - C:/Python34/circle.py (3.4.2)
File Edit Format Run Options Windows Help

```
import math

radius = int(input ("Enter the Radius: "))

pi = 22/7

Area_circle = pi * math.sqrt (radius)

print ("area of the circle is ", Area_circle, "square cms")
```

System analysis

- **A computer or information system is a collection of interrelated components that work together to process and provide information.**

- **An information/computer system constitutes users of the system, hardware, software, data, and a communication network.**

- **Users are required for the operation of any information system. These include:**
- **End users; these include those who feed the system with data and those who use the information produced. E.g. Accounts, secretaries, students customers, managers, etc.**

- **Hardware resources of a system include all physical devices such as video screen, computers, cameras, etc. and storage media.**

- **Software resources of a system include systems software such as the i) operating sytems, ii)application software such as an examination analysis program or accounting program, iii) the procedures, which are the operating instructions for people who will use an information system.**

- **Another type of system users are the information system specialists who are the people who develop and operated the system e.g. Information analysts, operators and programmers etc.**

- **The function of a system is to receive inputs and transform them into outputs.**

e.g. And information system accepts data as input and processes it into information.

- **System analysis is a stage in a system development cycle of an enquiry of the problem that an organisation will try to solve with an information system, which involves defining the problem, identifying its cause, specifying the solution and identifying the information requirements that must be met by a system solution.**

- **System development involves the activities that go into producing an information system's solution to an organisational problem. It includes defining, designing, testing and implementing a new system.**

- A system development life cycle includes the steps taken in the development of an information system. The steps include; problem identification, feasibility study, system analysis, system design, system coding, system implementation, system evaluation and maintenance.

Importance of systems analysis

- To overcome a high failure rate of the existing system
- The current system may be slow
- To address the complaints from clients of the organisation
- To address decline in profits or performance
- To reduce costs of operation and maintenance

- To meet the requirements of new technologies.
- To increase the flexibility of the current system

Phases of systems analysis